Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 335–352
Neural Networks
www.elsevier.com/locate/neunet

2007 Special Issue

Optimization and applications of echo state networks with leaky-integrator neurons

Herbert Jaeger[a,*], Mantas Lukoševičius[a], Dan Popovici[a], Udo Siewert[b]

[a] *Jacobs University Bremen, School of Engineering and Science, 28759 Bremen, Germany*
[b] *Planet intelligent systems GmbH, Residence Park 1-7, D-19065 Raben Steinfeld, Germany*

Abstract

Standard echo state networks (ESNs) are built from simple additive units with a sigmoid activation function. Here we investigate ESNs whose reservoir units are leaky integrator units. Units of this type have individual state dynamics, which can be exploited in various ways to accommodate the network to the temporal characteristics of a learning task. We present stability conditions, introduce and investigate a stochastic gradient descent method for the optimization of the global learning parameters (input and output feedback scalings, leaking rate, spectral radius) and demonstrate the usefulness of leaky-integrator ESNs for (i) learning very slow dynamic systems and replaying the learnt system at different speeds, (ii) classifying relatively slow and noisy time series (the Japanese Vowel dataset — here we obtain a zero test error rate), and (iii) recognizing strongly time-warped dynamic patterns.

Keywords: Recurrent neural networks; Pattern generation; Speaker classification

1. Introduction

The idea that gave birth to the twin pair of echo state networks (ESNs) (Jaeger, 2001) and liquid state machines (LSMs) (Maass, Natschlaeger, & Markram, 2002) is simple. Use a large, random, recurrent neural network as an excitable medium – the "reservoir" or "liquid", – which under the influence of input signals $u(t)$ creates a high-dimensional collection of nonlinearly transformed versions $x_i(t)$ – the activations of its neurons – of $u(t)$, from which a desired output signal $y(t)$ can be combined. This simple idea leads to likewise simple offline (Jaeger, 2001) and online (Jaeger, 2003) learning algorithms, sometimes amazingly accurate models (Jaeger & Haas, 2004), and may also be realized in vertebrate brains (Mauk & Buonomano, 2004; Stanley, Li, & Dan, 1999).

It is still largely unknown what properties of the reservoir are responsible for which strengths or weaknesses of an ESN for a particular task. Clearly, reservoirs differing in size, connectivity structure, type of neuron or other characteristics will behave differently when put to different learning tasks. A closer analytical investigation and/or optimization schemes of reservoir dynamics has attracted the attention of several authors (Ozturk, Xu & Principe, 2007; Schiller & Steil, 2005; Schmidhuber, Gomez, Wierstra, & Gagliolo, 2007; Zant, Becanovic, Ishii, Kobialka, & Plöger, 2004). A door-opener result for a deeper understanding of reservoirs/liquids in our view is the work of Maass, Joshi, and Sontag (in press) who show that LSMs with possibly nonlinear output readout functions can approximate dynamic systems of nth order arbitrarily well, if the liquid is augmented by n additional units which are trained on suitable auxiliary signals. Finally, it deserves to be mentioned that in theoretical neuroscience the question of how biological networks can process temporal information has been approached in a fashion that is related in spirit to ESNs/LSMs. Precise timing phenomena can be explained as emerging from the network dynamics as such, without the necessity of special timing mechanisms like clocks or delay lines (Mauk & Buonomano, 2004). Buonomano (2005)

* Corresponding author. Tel.: +49 421 200 3215.
E-mail addresses: h.jaeger@jacobs-university.de (H. Jaeger), m.lukosevicius@jacobs-university.de (M. Lukoševičius), d.popovici@jacobs-university.de (D. Popovici), siewert@planet.de (U. Siewert).
URLs: http://www.jacobs-university.de/ (H. Jaeger, M. Lukoševičius, D. Popovici), http://www.planet.de/ (U. Siewert).

0893-6080/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.04.016

presents an unsupervised learning rule for randomly connected, spiking neural networks that results in the emergence of neurons representing a continuum of differently timed stimulus responses, while preserving global network stability.

In this paper we add to this growing body of "reservoir research" and take a closer look at ESNs whose reservoir is made from leaky- integrator neurons. Leaky-integrator ESNs were in passing introduced in Jaeger (2001, 2002b); fragments of what will be reported here appeared first in a technical report (Lukoševičius, Popovici, Jaeger, & Siewert, 2006).

This article is composed as follows. In Section 2 we provide the system equations and point out basic stability conditions — amounting to algebraic criteria for the *echo state property* (Jaeger, 2001) in leaky-integrator ESNs. Leaky-integrator ESNs have one more global control parameter than the standard sigmoid unit ESNs have: in addition to the input and output feedback scaling, and the spectral radius of the reservoir weight matrix, a leaking rate has to be optimized. Section 3 explores the impact of these global controls on learning performance and introduces a stochastic gradient descent method for finding the optimal settings. The remainder is devoted to three case studies. First, managing very slow timescales by adjusting the leaky neurons' time constants is demonstrated with the "figure eight" problem (Section 4). This is an autonomous pattern generation task which also presents interesting dynamic stability challenges. Second, we treat the "Japanese Vowel" dataset. Using leaky- integrator neurons and some tricks of the trade we were able to achieve for the first time a zero test misclassification rate on this benchmark (Section 5). Finally, in Section 6 we demonstrate how leaky integrator ESNs can be designed which are inherently time-warping invariant.

For all computations reported in this article we used Matlab. The Matlab code concerning the global parameter optimization method, the "figure eight" and the Japanese Vowel studies is available online at http://www.faculty.iu-bremen.de/hjaeger/pubs.html.

2. Basic mathematical properties

2.1. System equations

We consider ESNs with K inputs, N reservoir neurons and L output neurons. Let $\mathbf{u} = \mathbf{u}(t)$ denote the K-dimensional external input, $\mathbf{x} = \mathbf{x}(t)$ the N-dimensional reservoir activation state, $\mathbf{y} = \mathbf{y}(t)$ the L-dimensional output vector, $\mathbf{W}^{\text{in}}$, $\mathbf{W}$, $\mathbf{W}^{\text{out}}$ and $\mathbf{W}^{\text{fb}}$ the input/internal/output/output feedback connection weight matrices of sizes $N \times K$, $N \times N$, $L \times (K + N)$ and $N \times L$, respectively. Then the continuous-time dynamics of a leaky-integrator ESN is given by

$$\dot{\mathbf{x}} = \frac{1}{c}\left(-a\,\mathbf{x} + f(\mathbf{W}^{\text{in}}\mathbf{u} + \mathbf{W}\mathbf{x} + \mathbf{W}^{\text{fb}}\mathbf{y})\right), \tag{1}$$

$$\mathbf{y} = g(\mathbf{W}^{\text{out}}[\mathbf{x}; \mathbf{u}]), \tag{2}$$

where $c > 0$ is a time constant global to the ESN, $a > 0$ is the reservoir neuron's leaking rate (we assume a uniform leaking rate for simplicity), f is a sigmoid function (we will use tanh), g is the output activation function (usually the identity or a sigmoid) and [;] denotes vector concatenation. Using an Euler discretization with stepsize δ of this equation we obtain the following discrete network update equation for dealing with a given discrete-time sampled input $\mathbf{u}(n\,\delta)$:

$$\mathbf{x}(n+1) = \left(1 - \frac{a\delta}{c}\right)\mathbf{x}(n) + \frac{\delta}{c} f(\mathbf{W}^{\text{in}}\mathbf{u}((n+1)\delta) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{\text{fb}}\mathbf{y}(n)), \tag{3}$$

$$\mathbf{y}(n) = g(\mathbf{W}^{\text{out}}[\mathbf{x}(n); \mathbf{u}(n\delta)]). \tag{4}$$

2.2. Stability: The echo state property

The Euler discretization leads to a faithful rendering of the continuous-time system (1) only for small δ. When δ becomes too large, the discrete approximation deteriorates or even can become unstable. There are several notions of stability which are relevant for ESNs. In Section 4 we will be concerned with the attractor stability of ESNs trained as pattern generators; but here we will start with a more basic stability property of ESNs, namely the *echo state property*. Several equivalent formulations of this property are given in Jaeger (2001). According to one of these formulations, an ESN has the echo state property if it washes out initial conditions at a rate that is independent of the input, for any input sequence that comes from a compact value set.

Definition 1. An ESN with reservoir states $\mathbf{x}(n)$ has the echo state property if for any compact $C \subset \mathbb{R}^K$, there exists a null sequence $(\delta_h)_{h=0,1,2,\ldots}$ such that for any input sequence $(\mathbf{u}(n))_{n=0,1,2,\ldots} \subseteq C$ it holds that $\|\mathbf{x}(h) - \mathbf{x}'(h)\| \leq \delta_h$ for any two starting states $\mathbf{x}(0)$, $\mathbf{x}'(0)$ and $h \geq 0$.

For leaky integrator ESNs, a sufficient and a necessary condition for the echo-state property are known, which we cite here from an early techreport (Jaeger, 2001).

Proposition 1. *Assume a leaky integrator ESN according to Eq. (3), where the sigmoid f is the* tanh *function and* (i) *the output activation function g is bounded (for instance, it is* tanh*), or* (ii) *there are no output feedbacks, that is, $\mathbf{W}^{\text{fb}} = \mathbf{0}$. Let $\sigma_{\max}$ be the maximal singular value of $\mathbf{W}$. Then if $|1 - \frac{\delta}{c}(a - \sigma_{\max})| < 1$ (where $\sigma_{\max}$ is the largest singular value of $\mathbf{W}$), the ESN has the echo-state property.*

The proof (a streamlined version of the proof given in Jaeger (2001) is given in the Appendix. A tighter sufficient condition for the echo state property in standard sigmoid ESNs has been given in Buehner and Young (2006); it remains to be transferred to leaky integrator ESNs.

Proposition 2. *Assume a leaky integrator ESN according to Eq. (3), where the sigmoid f is the* tanh *function. Then if the matrix $\tilde{\mathbf{W}} = \frac{\delta}{c}\mathbf{W} + (1 - a\frac{\delta}{c})\mathbf{I}$ (where $\mathbf{I}$ is the identity matrix) has a spectral radius $|\lambda|_{\max}$ exceeding 1, the ESN does not have the echo state property.*

The proof is a straightforward demonstration that the linearized ESN with zero input is instable around the zero state when $|\lambda|_{\max} > 1$, see Jaeger (2001). In practice it has always been

sufficient to ensure the necessary condition $|\lambda|_{\max}(\tilde{\mathbf{W}}) < 1$ for obtaining stable leaky integrator ESNs. We call the quantity $|\lambda|_{\max}(\tilde{\mathbf{W}})$ the *effective spectral radius* of a leaky integrator neuron.

One further natural constraint on system parameters is imposed by the intuitions behind the concept of leakage

$$a\,\frac{\delta}{c} \leq 1, \tag{5}$$

since a neuron should not in a single update leak more excitation than it has.

3. Optimizing the global parameters

In this section we discuss practical issues around the optimization of the various global parameters that occur in (3). By "optimization" we mainly refer to the goal of achieving a minimal training error. Achieving a minimal test error is delegated to cross-validation schemes which need a method for minimizing the training error as a substep.

We first observe that optimizing δ is by and large a non-issue. Raw training data will almost always be available in a discrete-time version with a given sampling period δ_0. Changing this given δ_0 means over- or subsampling. Oversampling might be indicated only in special cases, for instance, when the given data are noisy *and* when δ_0 is rather coarse *and* when a guided guess of the noiseless form of $\mathbf{u}(t)$ is available — then one may use these intuitions to first smoothe the noise out of $\mathbf{u}(t)$ with a tailored interpolation scheme, and then oversample to escape from the coarseness of δ_0; an example where this is common practice is the well-known laser dataset from the Santa Fé competition. A more frequently considered option will be subsampling with the aim of saving computation time in training and model exploitation. Opting for $\delta_1 = k\delta_0$, a network updated by (3) with δ_1 will generate in the n-th update cycle a reservoir state $\mathbf{x}(n)$ which will be close to the state $\mathbf{x}(kn)$ of a network updated with δ_0 — at least as long as the coarsening from δ_0 to δ_1 does not discard valuable information from the inputs, and as long as the coarser discretization from (1) to (3) does not incur a significant discretization error. Since the learnt output weights depend only on the pairings of reservoir states with teacher outputs, they will be similar in both cases. Thus, the question whether one subsamples is mainly a question of computational resource management and of whether no (or only negligible) relevant information from the training data gets lost. Beyond this, the quality of the ESN model should remain unaffected. We will therefore assume in the following that a suitable δ has been fixed beforehand, and we will write $\mathbf{u}(n)$ instead of $\mathbf{u}(n\delta)$ to indicate that the input sequence is considered a given, purely discrete sequence. This allows us to condense δ/c into a compound gain γ, giving

$$\begin{aligned}\mathbf{x}(n+1) \;=\; & (1-a\gamma)\,\mathbf{x}(n) + \gamma\, f(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) \\ & + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{\text{fb}}\mathbf{y}(n)),\end{aligned} \tag{6}$$

$$\mathbf{y}(n) = g(\mathbf{W}^{\text{out}}[\mathbf{x}(n); \mathbf{u}(n)]) \tag{7}$$

as a basis for our further considerations. Next we find that γ need not be optimized. For every ESN set up according to (6) and (7) there exists an alternative ESN with exactly the same output, which has $\gamma = 1$. This can be seen as follows. Let $\mathcal{E}$ be an ESN with weights $\mathbf{W}^{\text{in}}$, $\mathbf{W}$, $\mathbf{W}^{\text{fb}}$, leaking rate a and gain γ. Dividing both sides of (6) by γ, introducing $\tilde{a} = a\gamma$ and rewriting $\mathbf{W}\mathbf{x}(n)$ to $(\gamma\mathbf{W})\frac{1}{\gamma}\mathbf{x}(n)$ gives

$$\begin{aligned}\frac{1}{\gamma}\mathbf{x}(n+1) = (1-\tilde{a})\,\frac{1}{\gamma}\mathbf{x}(n) + f\Big(&\mathbf{W}^{\text{in}}\mathbf{u}(n+1) \\ &+ (\gamma\mathbf{W})\frac{1}{\gamma}\mathbf{x}(n) + \mathbf{W}^{\text{fb}}\mathbf{y}(n)\Big),\end{aligned} \tag{8}$$

which gives the discrete update dynamics for an ESN $\tilde{\mathcal{E}}$ having internal weights $\tilde{\mathbf{W}} = \gamma\mathbf{W}$ and $\tilde{\gamma} = 1$, such that this dynamics is identical to the update dynamics of $\mathcal{E}$ except for a scaling factor $1/\gamma$ of the states. If we scale the first N components of $\mathbf{W}^{\text{out}}$ by γ, obtaining $\tilde{\mathbf{W}}^{\text{out}}$, the resulting output sequence

$$\mathbf{y}(n) = g\left(\tilde{\mathbf{W}}^{\text{out}}\left[\frac{1}{\gamma}\mathbf{x}(n); \mathbf{u}(n)\right]\right) \tag{9}$$

is identical to that from (4). Thus we may without loss of generality use the update equation

$$\begin{aligned}\mathbf{x}(n+1) \;=\; & (1-a)\,\mathbf{x}(n) + f(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) \\ & + (\gamma\mathbf{W})\mathbf{x}(n) + \mathbf{W}^{\text{fb}}\mathbf{y}(n)),\end{aligned} \tag{10}$$

or equivalently, if we agree that $\mathbf{W}$ have unit spectral radius and that the input and feedback weight matrices $\mathbf{W}^{\text{in}}$, $\mathbf{W}^{\text{fb}}$ are normalized to entries of unit maximal absolute size,

$$\begin{aligned}\mathbf{x}(n+1) \;=\; & (1-a)\,\mathbf{x}(n) + f(s^{\text{in}}\mathbf{W}^{\text{in}}\mathbf{u}(n+1) \\ & + (\rho\mathbf{W})\mathbf{x}(n) + s^{\text{fb}}\mathbf{W}^{\text{fb}}\mathbf{y}(n) + s^{\nu}\nu(n+1)),\end{aligned} \tag{11}$$

$$\mathbf{y}(n) = g(\mathbf{W}^{\text{out}}[\mathbf{x}(n)\;;\;\mathbf{u}(n)]) \tag{12}$$

where ρ is now the spectral radius of the reservoir weight matrix and s^{in}, s^{fb} are the scalings of the input and the output feedback. We have also added a state noise term ν in (11), where we assume that $\nu(n+1)$ is a suitably normalized noise vector (we use uniform noise ranging in $[-0.5, 0.5]$, but Gaussian noise with unit variance would be another natural choice). The scalar s^{ν} scales the noise. This noise term is going to play an important role. Eqs. (11) and (12) establish the platform for all our further investigations. Considering these equations, we are faced with the task to optimize six global parameters for minimal training error: (i) reservoir size N, (ii) input scaling s^{in}, (iii) output feedback scaling s^{fb}, (iv) reservoir weight matrix spectral radius ρ, (v) leaking rate a, and (vi) noise scaling s^{ν}.

Stability of ESNs may become problematic when there is output feedback. Output feedback is mandatory for pattern-generating ESNs, and we will be confronted with this issue in Section 4. The bulk of RNN applications, however, is nongenerative but purely input-driven (e.g. dynamic pattern recognition, time series prediction, filtering or control), and ESN models for such applications should, according to our experience, be set up without output feedback.

Among these six global parameters, the reservoir size N plays a role which is different from the roles of the others. Optimizing N means to negotiate a compromise between model

bias and model variance. The other global controls shape the dynamical properties of the ESN reservoir. One could say, N determines the model *capacity*, and the others the model *characteristic*. With respect to N, we usually start out from the rule of thumb that the N should be about one tenth of the length of the training sequence. If the training data are deterministic or come from a very low-noise process, we feel comfortable with using larger N — for instance, when learning to predict a chaotic process from almost noise-free training data (Jaeger & Haas, 2004), we used a 1000 unit reservoir with 2000 time steps of training data. Fine-tuning N is then done by cross-validation.

Having decided on N in this roundabout fashion, we proceed to optimize the parameters (ii)–(vi). This is a nonlinear, five-dimensional minimization problem, where the target function is the training error. In the past no informed, automated search method was available, so we always tuned the global learning controls by manual search. This works to the extent that the experimenter has good intuitions about how the reservoir dynamics is shaped by these controls. In our experience with student projects, it became clear that persons lacking a long-time experience with ESNs often settle on quite suboptimal global controls. Furthermore, in online adaptive usages of ESNs an automated optimization scheme would also be welcome.

We now develop such a method. It will be based on a stochastic gradient descent, where the controls (ii)–(v) follow the downhill gradient of the training error. The noise level s^ν will play an important role in stabilizing this gradient descent; it is not itself a target of optimization. To set the stage, we first recapitulate the stochastic gradient optimization of the output weights with the least mean square (LMS) algorithm (Farhang-Boroujeny, 1998) known from linear signal processing. Let $(\mathbf{u}(n), \mathbf{d}(n))_{n\geq 1}$ be a right-infinite signal with input $\mathbf{u}(n)$ and a (teacher) output $\mathbf{d}(n)$. The objective is to adapt the output weights $\mathbf{W}^{\text{out}}$ online using time-dependent weights $\mathbf{W}^{\text{out}}(n)$ such that the network output $\mathbf{y}(n)$ follows $\mathbf{d}(n)$. Introducing the error and squared error

$$\boldsymbol{\epsilon}(n) = \mathbf{d}(n) - \mathbf{y}(n), \qquad E(n) = \frac{1}{2}\,\|\boldsymbol{\epsilon}(n)\|^2. \tag{13}$$

the LMS rule adapts the output weights by

$$\mathbf{W}^{\text{out}}(n+1) = \mathbf{W}^{\text{out}}(n) + \lambda\,\boldsymbol{\epsilon}(n)[\mathbf{x}(n-1); \mathbf{u}(n)]^{\mathsf{T}}, \tag{14}$$

where λ is a small adaptation rate and $\cdot^{\mathsf{T}}$ denotes the vector/matrix transpose. This equation builds on the fact that

$$\frac{\partial E(n+1)}{\partial \mathbf{W}^{\text{out}}} = \boldsymbol{\epsilon}(n+1)[\mathbf{x}(n-1); \mathbf{u}(n)]^{\mathsf{T}}, \tag{15}$$

so the update (14) can be understood as an instantaneous attempt to reduce the current output error by adapting the weights.

In order to optimize the global controls a, ρ, s^{in} and s^{fb} by a similar stochastic gradient descent, we must calculate $\partial E(n+1)/\partial p$, where p is one of a, ρ, s^{in}, or s^{fb}.

Using $\mathbf{0}_{\mathbf{u}} = (0, \ldots, 0)^{\mathsf{T}}$ (where $\cdot^{\mathsf{T}}$ denotes vector/matrix transpose; as many zeros as there are input dimensions) for a shorter notation, the chain rule yields for all $p \in \{a, \rho, s^{\text{in}}, s^{\text{fb}}\}$

$$\frac{\partial E(n+1)}{\partial p} = -\boldsymbol{\epsilon}^{\mathsf{T}}(n+1)\mathbf{W}^{\text{out}}(n+1)\left[\frac{\partial \mathbf{x}(n+1)}{\partial p}\;;\;\mathbf{0}_{\mathbf{u}}\right], \tag{16}$$

Thus we have to compute the various $\partial \mathbf{x}(n)/\partial p$. Again invoking the chain rule, and observing (12) (where we assume linear output units with $g = \mathbf{id}$), and putting $X(n) = s^{\text{in}}\mathbf{W}^{\text{in}}\mathbf{u}(n) + (\rho\mathbf{W})\mathbf{x}(n-1) + s^{\text{fb}}\mathbf{W}^{\text{fb}}\mathbf{y}(n-1)$, we obtain

$$\begin{aligned}\frac{\partial \mathbf{x}(n)}{\partial a} &= (1-a)\,\frac{\partial \mathbf{x}(n-1)}{\partial a} - \mathbf{x}(n-1) + f'(X(n)) \\ &\quad .*\left(\rho\mathbf{W}\frac{\partial \mathbf{x}(n-1)}{\partial a} + s^{\text{fb}}\mathbf{W}^{\text{fb}}\mathbf{W}^{\text{out}}(n-1)\right. \\ &\quad \left.\times\left[\frac{\partial \mathbf{x}(n-2)}{\partial a}\;;\;\mathbf{0}_{\mathbf{u}}\right]\right) \end{aligned} \tag{17}$$

$$\begin{aligned}\frac{\partial \mathbf{x}(n)}{\partial \rho} &= (1-a)\,\frac{\partial \mathbf{x}(n-1)}{\partial \rho} + f'(X(n)) \\ &\quad .*\left(\rho\mathbf{W}\frac{\partial \mathbf{x}(n-1)}{\partial \rho} + \mathbf{W}\mathbf{x}(n-1)\right. \\ &\quad \left. + s^{\text{fb}}\mathbf{W}^{\text{fb}}\mathbf{W}^{\text{out}}(n-1)\left[\frac{\partial \mathbf{x}(n-2)}{\partial \rho}\;;\;\mathbf{0}_{\mathbf{u}}\right]\right) \end{aligned} \tag{18}$$

$$\begin{aligned}\frac{\partial \mathbf{x}(n)}{\partial s^{\text{in}}} &= (1-a)\,\frac{\partial \mathbf{x}(n)}{\partial s^{\text{in}}} + f'(X(n)) \\ &\quad .*\left(\rho\mathbf{W}\frac{\partial \mathbf{x}(n-1)}{\partial s^{\text{in}}} + \mathbf{W}^{\text{in}}\mathbf{u}(n)\right. \\ &\quad \left. + s^{\text{fb}}\mathbf{W}^{\text{fb}}\mathbf{W}^{\text{out}}(n-1)\left[\frac{\partial \mathbf{x}(n-2)}{\partial s^{\text{in}}}\;;\;\mathbf{0}_{\mathbf{u}}\right]\right) \end{aligned} \tag{19}$$

$$\begin{aligned}\frac{\partial \mathbf{x}(n)}{\partial s^{\text{fb}}} &= (1-a)\,\frac{\partial \mathbf{x}(n-1)}{\partial s^{\text{fb}}} + f'(X(n)) \\ &\quad .*\left(\rho\mathbf{W}\frac{\partial \mathbf{x}(n-1)}{\partial s^{\text{fb}}} + \mathbf{W}^{\text{fb}}y(n-1)\right. \\ &\quad \left. + s^{\text{fb}}\mathbf{W}^{\text{fb}}\mathbf{W}^{\text{out}}(n-1)\left[\frac{\partial \mathbf{x}(n-2)}{\partial s^{\text{fb}}}\;;\;\mathbf{0}_{\mathbf{u}}\right]\right) \end{aligned} \tag{20}$$

where $.*$ denotes component-wise multiplication of two vectors. These equations simplify when there is no output feedback.

$$\begin{aligned}\frac{\partial \mathbf{x}(n)}{\partial a} &= (1-a)\,\frac{\partial \mathbf{x}(n-1)}{\partial a} - \mathbf{x}(n-1) + f'(X(n)) \\ &\quad .*\left(\rho\mathbf{W}\frac{\partial \mathbf{x}(n-1)}{\partial a}\right) \end{aligned} \tag{21}$$

$$\begin{aligned}\frac{\partial \mathbf{x}(n)}{\partial \rho} &= (1-a)\,\frac{\partial \mathbf{x}(n-1)}{\partial \rho} + f'(X(n)) \\ &\quad .*\left(\rho\mathbf{W}\frac{\partial \mathbf{x}(n-1)}{\partial \rho} + \mathbf{W}\mathbf{x}(n-1)\right) \end{aligned} \tag{22}$$

$$\begin{aligned}\frac{\partial \mathbf{x}(n)}{\partial s^{\text{in}}} &= (1-a)\,\frac{\partial \mathbf{x}(n)}{\partial s^{\text{in}}} + f'(X(n)) \\ &\quad .*\left(\rho\mathbf{W}\frac{\partial \mathbf{x}(n-1)}{\partial s^{\text{in}}} + \mathbf{W}^{\text{in}}\mathbf{u}(n)\right). \end{aligned} \tag{23}$$

Eqs. (17)–(20) (or (21)–(23) when there are no output feedbacks) provide a recursive scheme to compute $\partial \mathbf{x}(n)/\partial p$

from $\partial \mathbf{x}(n-1)/\partial p$ and $\partial \mathbf{x}(n-2)/\partial p$, or respectively from $\partial \mathbf{x}(n-1)/\partial p$ only in the absence of output feedback. At startup times 1 and 2, the missing derivatives are initialized by arbitrary values (we set them to zero).

Inserting the expressions $\partial \mathbf{x}(n)/\partial p$ into (16) finally gives us the desired (recursive) stochastic parameter updates:

$$p(n+1) = p(n) - \kappa \frac{\partial E(n+1)}{\partial p}, \tag{24}$$

where κ is an adaptation rate. We remark in passing that when the tanh is used for the activation function f, its derivative is given by

$$f'(x) = \tanh'(x) = \frac{4}{2 + \exp(2x) + \exp(-2x)}. \tag{25}$$

One caveat is that the parameter updates must be prevented to lead the ESN away from the echo state property. Specifically, note that ρ and a together determine the effective spectral radius $|\lambda|_{\max}$ of the reservoir, which is the spectral radius of the matrix $\rho\mathbf{W} + (1-a)\mathbf{I}$ (Proposition 2; recall that we agreed here on $\mathbf{W}$ having unit spectral radius). From experience we know that the echo state property is given if $|\lambda|_{\max} < 1$. A simple algebra argument shows that $|\lambda|_{\max} \leq \rho + 1 - a$; the bound tightens to an equality iff the largest eigenvalue of $\mathbf{W}$ is its spectral radius. Thus the echo-state property is for all practical purposes guaranteed when

$$\rho \leq a. \tag{26}$$

This is easy to enforce while the online adaptation proceeds. We proceed to explain the practical usage of our stochastic update equations. First note that one has to carry out a simultaneous optimization of the output weights $\mathbf{W}^{\text{out}}$ and the controls (ii)–(v), i.e. perform an error gradient descent in the combined output weight and control parameter space. We have tested two variants:

A. *Stochastic output weight update*. The updates (24) are executed together with the stochastic output weight updates (14). Whether in a time step one first updates the weights and then the parameters, or vice versa, should not matter. We updated the weights second, but haven't tried otherwise.
B. *Pseudo-batch output weight update*. The output weights are updated much more rarely than the control parameters, which are updated at every time step using (24). At every T-th cycle, the output weights are recomputed with the known batch learning algorithm. Specifically, at times $m = T\ n + 1, \ldots, (T+1)\ n$ harvest the states $[\mathbf{x}(m)\ ;\ \mathbf{u}(m)]$ row-wise into a $T \times (N+K)$ buffer matrix A and the teacher outputs $d(m)$ into a $T \times 1$ matrix B. At time $(T+1)\ n$ compute new output weights by linear regression of the teachers on the states via $\mathbf{W}^{\text{out}}((T+1)n) = (A^+\ B)^{\mathsf{T}}$. When state noise $v(n)$ is used, we added it to A after harvesting the states, which are, however, updated without noise in order not to disturb the adaptation of the parameters p.

We found that method B gave faster convergence with an increased risk of instability and generally preferred it. Method A may be the better choice in automated online adaptation applications.

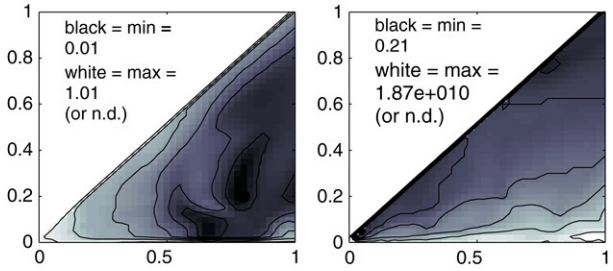

Fig. 1. NRMSE (left) and absolute mean output weights (right) obtained in the multiple sines recall task with different values of a (x-axis) and ρ (y-axis). The shades of grey represent the decadic logarithm of the NRMSEs and weight sizes, such that black corresponds to the minimal value in the diagram and white to the maximal value. The upper triangle in each panel is blank because here the effective spectral radius exceeds one, and no model was trained. For details see text.

To initialize the procedure, we fixed some reasonable starting values $p(0)$ for our four global parameters and computed initial output weights $\mathbf{W}^{\text{out}}(0)$ by the standard batch method, using controls $p(0)$ and a short training subsequence.

In our numerical experiments, certain difficulties became apparent which shed an interesting light on ESNs in general. For the sake of discussion, let us consider a concrete task. A one-dimensional input sequence $u(n) = \sin(n) + \sin(0.51*n) + \sin(0.22*n) + \sin(0.1002*n) + \sin(0.05343*n)$ was prepared, a sum of essentially incommensurate sines with periods ranging from about 6 to about 120 discrete time steps. This sequence was scaled to a range in [0, 1]. The desired output was $d(n) = u(n-5)$, that is, the task was a delay-5, short-term recall task. A small ESN with $N = 10$ was employed. Due to its sigmoid units in the reservoir, such a small ESN would not be able to recall with delay-5, a white noise signal; for any reasonable performance on the multiple sines input the network has to exploit the regularity in the data.

To get a first impression of the impact of global control parameters on the training error, we fixed s^{in} at a value of 0.3, and trained the ESN offline with zero state noise ($s^{\nu} = 0$) on a 500-step training sequence (plus 200-steps initial washout). We did this exhaustively for all values of $0 \leq \rho \leq a \leq 1$ in increments of 1/40. Fig. 1 shows the NRMSEs and the mean absolute output weight sizes obtained in each cell in the 41×41 grid of this a–ρ-cross-section in parameter space.

Several features of these plots deserve a comment. First, the NRMSE landscape in this very simple task has at least three local minima. Second, these minima are obtained at values for $a < 1$, that is, the leaky integration mechanism is, in fact, useful. Third, the best NRMSEs fall into regions of large output weights (order of 1000 and above). Large output weights are as a rule undesirable for ESNs. They typically imply a poor generalization to test data unless the latter come from exactly the same (deterministic) source as the training data.

Large output weights – or rather, the fragility of model performance in the face of perturbations which they usually imply – also are prone to create difficulties for a gradient descent approach to optimize the global controls p. The danger

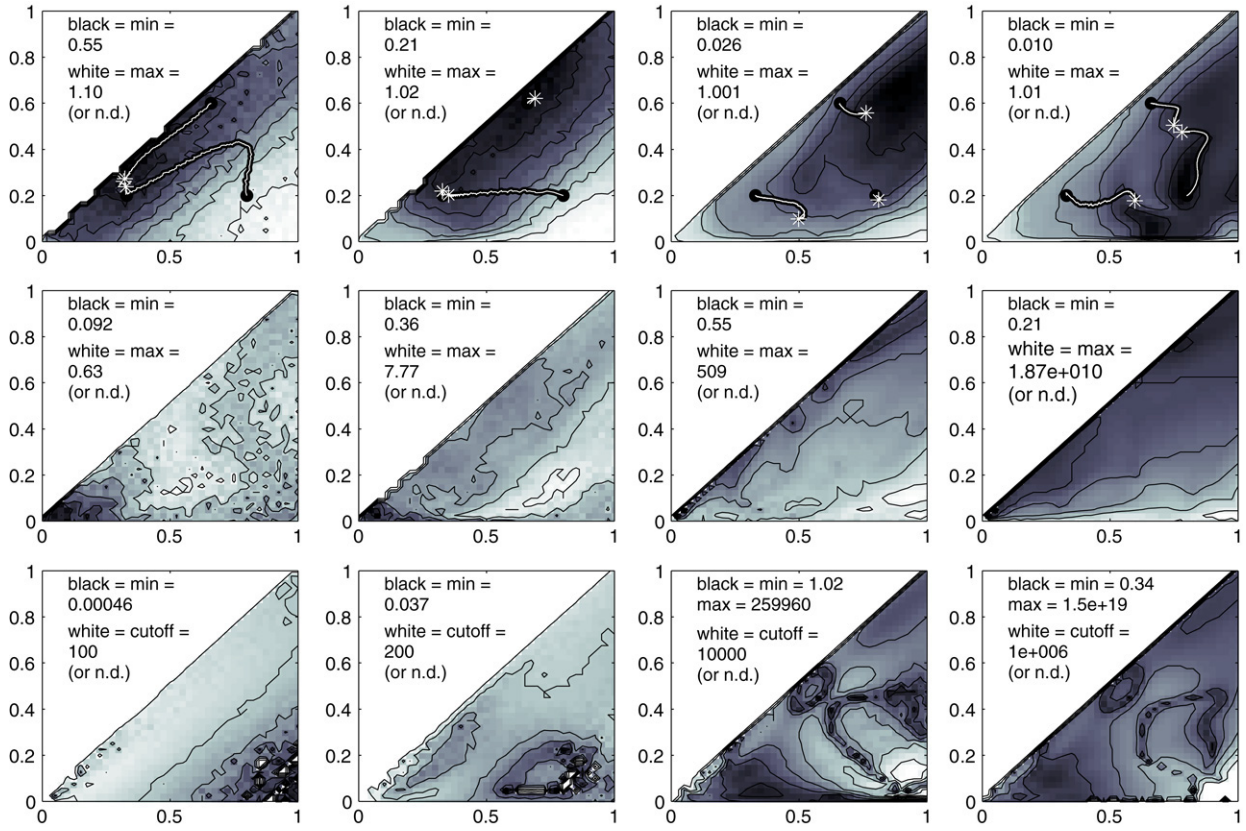

Fig. 2. The multiple Sines recall task. In each panel, the x-axis is a and the y-axis is ρ. In each row of panels, from left to right the panels correspond to state noise levels of size 0.1, 0.01, 0.0001 and 0.0. Top row: NRMSE reliefs with traces of the gradient descent algorithm at work. Centre row: average absolute output weight sizes. Bottom row: curvatures $\partial^2 E/\partial a^2$. Tiny white speckles in some curvature plots result from rounding noise close to zero — should be black. Elevations in all surface plots are in logarithmic (base 10) scale. Several surface plots have clipped maximum height, see inset texts.

lies in the fact that small changes of p, given large output weights $\mathbf{W}^{\text{out}}(n)$, will likely lead to great changes in $E(n+1)$. In geometrical terms, the curvatures $\partial^2 E(n)/\partial p^2(n)$ is large and thereby dictates small adaptation rates κ to ensure stability.

To illustrate this, we fixed $s^{\text{in}}_{\text{ref}} = 0.3, s^{\text{fb}}_{\text{ref}} = 0$ and numerically estimated and plotted $\partial^2 E/\partial a^2$ for each of the cells in the same 41×41 a–ρ-grid as before. Fig. 2 (bottom row, rightmost panel) shows the result. Juxtaposing this graph with the NRMSE and output weight plots in the top and centre row above it in Fig. 2 we find that first, some of the a–ρ-combinations where the NRMSE attains a local minimum fall into a range where the output weights are greater than 1000 and the curvature is greater than 10,000, and second, many paths along the gradients in the NRMSE-landscape shown in Fig. 1 (left) will cross areas in the curvature landscape with values greater than 10,000. As a necessary condition for stability of gradient descent in the a direction, the adaptation rate must not exceed half the curvature, that is, $\kappa \leq 1/5000$. But since in some (small) regions in this a–ρ-cross-section the curvature attains values even greater than 1e+19, to ensure stability of a gradient descent in all circumstances we would have to reduce κ to less than 1e−19. This would result in an impossibly slow progress. A dynamic adaptation of the adaptation rate would not solve the problem, because the adaptation of κ would have to trim κ to microscopic size when passing through regions of very large curvature — the process would effectively get halted in these spots.

In the example from the rightmost column of panels in Fig. 2 we find that the correlation between the size of output weights and curvature is clearly apparent as a trend (but not everywhere consistent). Small output weights can be enforced by state noise administered during training or other regularization methods — all of which, however, may incur a loss in training and testing accuracy if high-precision training data are available. To illustrate the generally beneficial aspects of small output weights, we repeated the curvature survey in the a–ρ-cross-section with the same settings as before, but administering now state noise of sizes $s^{\nu} = 0.1;\ 0.01;\ 0.0001$ (first three panels in bottom row in Fig. 2). The curvature is thereby much reduced. The means of the curvature values in the bottom panels of Fig. 2 is 26.2 ($s^{\nu} = 0.1$); 107.4 ($s^{\nu} = 0.01$); 889.7 ($s^{\nu} = 0.0001$) and 1.4e+006 (zero noise) [we discarded the cells on the lower border of the grid for the computation of the means because the curvatures explode at the very margin where $\rho = 0$, an unnatural condition for an ESN]. Thus, our ESN updated with a state noise of size 0.1 would admit an adaptation rate five orders of magnitude larger than in the zero noise condition. This is of course only a coarse and partial account. But the basic message is clear: stability is a critical issue in this game, and it can be improved (and convergence sped up) by a noisy state update (or other regularizing methods that yield small output weights).

In order to assess the practicality of the proposed stochastic gradient descent method, and at the same time the impact of state noise on the model accuracy and the location of optimal parameters p, we ran the method from three starting values $a/\rho = 0.33/0.2; 0.8/0.2; 0.66/0.2$, in the version with no output feedback. To obtain meaningful graphical renderings of the results in the a–ρ-plane, we froze s^{in} at a value of 0.3, which was found to be close to the optimal value in preliminary experiments. We used the "B" version of output weight adaptation. We ran the method with four state noise levels of $s^{\nu} = 0.1; 0.01; 0.0001; 0.0$. The corresponding adaptation rates were $\kappa = 0.002; 0.002; 0.0002; 0.0001$, and the sampling periods T for the recomputation of the output weights were $T = 1000; 1000; 100; 50$. The reason for choosing smaller T in the lower-noise conditions is that the output weights have to keep track of the parameter adaptation more closely due to the greater curvature of the parameter error gradients. The method was run for a duration of 200,000; 200,000; 1,000,000; 1,000,000 steps in the four conditions.

The top row in Fig. 2 shows the resulting developments of $a(n)$ and $\rho(n)$. The centre row panels in the figure show the corresponding "weight scapes", i.e. the average absolute output weights obtained for the various a–ρ-combinations. Progress of gradient descent is much faster in the two high-noise conditions (mind the different runtimes of 200,000 vs. 1 Mio steps). Attempts to speed up the low-noise trials by increasing κ led to instability. We monitored the NRMSE development of each of the gradient traces (not shown here); they all were noisily decreasing (as it should be) except for the leftmost trace in the noise-0.0001-panel. When this particular trace takes its final right turn, away from what clearly would be the downhill direction, the NRMSE starts to increase and the output weights sizes start to fluctuate greatly — a sign of impending instability.

The stochastic developments of $a(n)$ and $\rho(n)$ in the top row of Fig. 2 clearly does not follow the gradient indicated by the grey scale landscape. The reason is that the grey shade level plots are computed by running the standard offline learning algorithm for ESNs for each grid cell on a fixed training sequence, whereas the gradient descent traces compute ESN output weights as they move along, via the method "B" described above, from reservoir states that have been collected while the parameter adaptation proceeded. That is, the output weights which the gradient descent method uses are derived from somewhat noisier reservoir states than the NRMSE level plots, because in addition to the administered state noise there was parameter adaptation noise (a remedy suggested by a reviewer would be to alternate between periods where parameters p are not adjusted and states without adaptation noise are collected for updating the output weights, and periods where parameters p are stochastically adjusted with fixed output weights — we did not try this because it would double the computational load). Furthermore, the output weight updates at intervals of T always "come late" with respect to the a and ρ updates, using reservoir states which were obtained with earlier a and ρ. Thus all we should expect to learn from these graphics is whether the asymptotic values of a and ρ coincide with the minima of the error landscape — which is the case for the high-noise condition but increasingly becomes false with reduced noise.

The compound theme of noise or other regularizers, stability, generalization capabilities, and size of output weights calls for further investigations. It is in our view one of the most important goals for ESN research to understand how reservoirs can be designed or pretrained, depending on the learning task, such that small output weights will be obtained *without* adding state noise or invoking other regularizers.

4. The lazy figure eight

In this section we will train a leaky integrator ESN to generate a slow "figure eight" pattern in two output neurons, and we will dynamically change the time constant in the ESN equations to slow down and speed up the generated pattern.

The "figure 8" generation task is a perennial exercise for RNNs (for example, see Pearlmutter (1995), Zegers and Sundareshan (2003) and references therein). The task appears not very complicated, because a "figure 8" can be interpreted as the superposition of a sine (for the x direction) and a cosine of half the sine's frequency (for the y direction). However, a closer inspection of this seemingly innocent task will reveal surprisingly challenging stability problems.

Since in this section we will be speeding up and slowing down a performing ESN, we use the network equation (6) as a basis. The gain γ will eventually be employed to set the "performance speed" of the trained ESN. Since our figure eight pattern will be generated on a very slow timescale, we will refer to it as the "lazy eight" (L8) task.

As a teacher signal, we used a discrete-time version of the L8 trajectory which is centered on the origin, upright, and scaled to fill the $[-1, 1]$ range in both coordinate directions. One full ride along the eight uses 200 sampling points. Thus this signal is periodic of length 200. Periodic signals of rugged shape and modest length (up to about 50 time points) have been trained on standard ESNs (Jaeger, 2001) with success, but longer, smoothly and slowly changing targets like a 200-point "lazy" eight we could never master with standard ESNs. While low training errors can easily be achieved, the trained nets are apparently always instable when they are run in autonomous patter generation mode. Here is one sufficient explanation of this phenomenon (there may be more). When a standard, sigmoid-unit reservoir is driven by a slow input signal (from the perspective of the reservoir, the feedback from the output just is a kind of input), it is "adiabatically" following a trajectory along fixed point attractors of its dynamics which are defined by the current input. If the input would freeze at a particular value, the network state would also freeze at the current values (reminiscent of the equilibrium point theory of biological motor control (Hinder & Milner, 2003).) This implies that such a network cannot cope with situations like in the center crossing of the figure 8, because, in sloppy terms, it can't remember where it came from and thus does not know where to go. Indeed, when one tries to make a sigmoid-unit network, which had been trained on figure eight data, to generate that trajectory

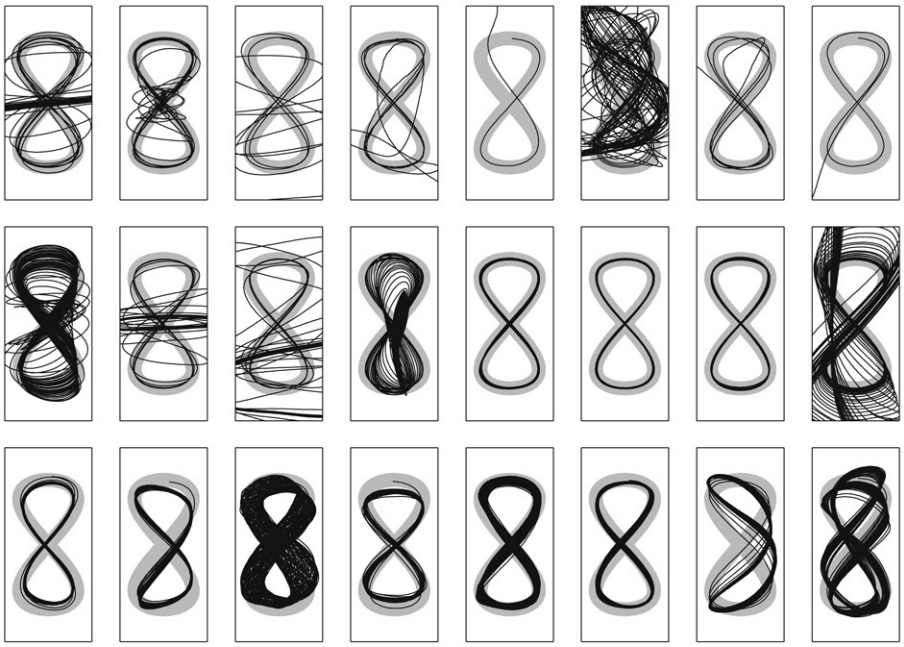

Fig. 3. Lazy 8 test performance. Top row: baseline study, no precautions against instability taken. Center row: output weights computed with a ridge regression regularizer. Bottom row: yield of the "noise immunization" method. Each panel shows a plot window of $[-1.5, 1.5]$ in x and y direction (stretched for plotting). The network-generated trajectory is shown as a solid black line; the gray figure 8 in the background marks the location of the teacher eight. The three plots in each column each resulted from the same ESN which was trained in different ways.

autonomously, it typically goes haywire immediately when it reaches the crossing point.

If in contrast we drive a leaky integrator reservoir that has a small gain γ with a slow input signal, its slow dynamics is not a forced equilibrium dynamics, but instead can memorize long-past history in its current state, so at least one reason for the failure of standard ESNs on such tasks is no longer relevant.

As an baseline study, we trained a leaky integrator ESN on the lazy eight, using $N = 20$, $\rho = 0.02$, $a = 0.02$, $\gamma = 1.0$, $s_{\text{fb}} = 1$. The motivation to use these settings is explained later. A constant bias input of size 1.0 was fed to the reservoir with $s_{\text{in}} = 0.1$. The reservoir weight matrix had a connectivity of 50%. Linear output units were used. The setting $\rho = a = 0.02$ implies a maximal effective spectral radius of $|\lambda|_{\max}(\tilde{\mathbf{W}}) = 1.0$ in cases where the spectral radius is equal to the largest eigenvalue. This, however, is rarely the case with randomly generated reservoirs, who typically featured an effective spectral radius between 0.993 and 0.996.

For training, a 3000 step two-dimensional sequence (covering 15 full swings around the figure eight) was used. In each trial, the various fixed weight matrices were randomly created with the standard scalings described before Eq. (11). The ESN was then driven by the training sequence, which was teacher-forced into the output units. The first 1000 network states were discarded to account for the initial transient and the remaining 2000 ones were used to compute output weights by linear regression. The trained ESN was then tested by first teacher-forcing it again for 1000 steps, after which it was decoupled from the teacher and run on its own for another 19,000 steps. During this autonomous run, a uniform state noise $\nu(n)$ of component amplitude 0.01 was added to the reservoir neurons to test whether the generated figure eight was dynamically stable (in the sense of being a periodic attractor). Fig. 3 (top row) shows the results of 8 such runs. None of them is stable — in fact, among 20 trials which were carried out, none resulted in a periodic attractor resembling the original figure eight.

At this point some general comments on the stability of trained periodic pattern generators are in place. First, we want to point out that the figure eight task – and its stability problem – is related to the "multiple superimposed oscillations" (MSO) pattern generation task, which recently has been variously investigated in an ESN/LSM context (for instance, in the contribution by Xue et al. (in this issue)). In the MSO task, a one-dimensional quasi-periodic signal made from the sum of a few incommensurate sines has to be generated by an ESN. Most observations that follow concerning the L8 task pertain to the MSO task as well.

Second, generating signals which are superpositions of sines is of course simple for *linear* dynamic systems. It presents no difficulty to train a linear ESN reservoir on the L8 or the MSO task; one will obtain training errors close to machine precision as long as one has at least two reservoir neurons per sine component. When the trained network is run as a generator, one will witness an extremely precise autonomous continuation of the target signal after the network has been started by a short teacher-forced phase. The impression of perfection is, however, deceptive. In a linear system, the generated sine components are not stably phase-coupled, and individually they are not asymptotically stable oscillations. In the presence of perturbations, both the amplitude and the relative phase of the sine components of the generated signal will go off on a random walk. This inherent lack of asymptotic stability is not what one would require from a pattern generation mechanism. In an

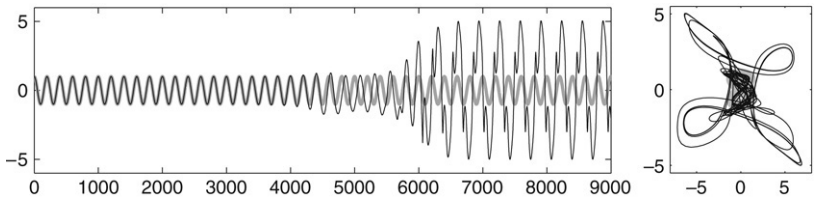

Fig. 4. Typical behaviour of naively trained L8 ESN models. The plot left shows one of the two output signals (thick grey line: correct; thin black: network production; x-axis: update cycles) in free-running generation mode. After an initial period where the network-produced signal is all but identical to the target, the system's instability becomes eventually manifest by a transition into an attractor unrelated to the task. Here the ultimate attractor has an appealing butterfly shape (panel right).

ESN context this implies that one has to use *nonlinear* reservoir dynamics (or, at least, nonlinear output units).

Third, when training nonlinear reservoir ESNs on the L8 or MSO task, one can still obtain extremely low training errors, and autonomous continuations of the target signal which initially look perfect. We confess that we have ourselves been fooled by ESN generated L8 trajectories which looked perfect for thousands of update cycles. However, when we tested our networks for longer timespans, the process invariably at some point went havoc. Fig. 4 gives an example. We suspect (and know in some cases) that a similarly deceptive initial well-behaviour may have led other investigators of RNN-based pattern generators to false positive beliefs about their network's performance.

Fourth, once one grows apprehensive of the stability issue, one starts to suspect that in fact here there is a rather nontrivial challenge. On the one hand, we *need* a nonlinear generating system to achieve asymptotic stability. On the other hand, we cannot straightforwardly devise of the nonlinear generator as a compound made of coupled nonlinear generators of the component sines. Coupling nonlinear oscillators of different frequencies is a delicate manoeuvre which is prone to spur interesting, but unwanted, dynamic side-effects. In other words, while the MSO and L8 signals can be seen as additive superpositions of sines (in a linear systems view), nonlinear oscillators do not lend themselves to noninterfering "additive" couplings.

In this situation, we investigated two strategies to obtain stable L8 generators. For one, following a suggestion from J. Steil we computed the output weights with a strong Tikhonov regularization (aka ridge regression, see Hastie, Tibshirani, and Friedman (2001)), using

$$\mathbf{W}^{\text{out}} = \left((\mathbf{S}^{\mathsf{T}}\mathbf{S} + \alpha^2 \mathbf{I})^{-1} \, \mathbf{S}^{\mathsf{T}}\mathbf{T} \right)^{\mathsf{T}}, \tag{27}$$

where $\mathbf{S}$ is the matrix containing the network-and-input vectors harvested during training in its rows, $\mathbf{T}$ contains the corresponding teacher vectors in its rows, and α is the regularization coefficient. For $\alpha = 0$ one gets a standard linear regression (by the Wiener–Hopf solution). We used a rather large value of $\alpha = 1$.

Our other strategy used noise to "immunize" the two component oscillators against interference with each other. According to this strategy, the output weights for the two output units were trained separately, each needing one run through the training data. For training the first output unit, a two-dimensional figure eight signal was prepared, which in the first component was just the original teacher $d_1(n)$, but in the second component was a mixture $(1 - q)\ d_2(n) + q\ \nu(n)$ of the correct second figure eight signal $d_2(n)$ with a noise signal $\nu(n)$. The noise signal $\nu(n)$ was created as a random permutation of the correct second signal and thus had an identical average amplitude composition. The resulting two-dimensional signal $(d_1(n), (1-q)\ d_2(n) + q\ \nu(n))^{\mathsf{T}}$ was teacher-forced on the two output units during the state harvesting period. The collected states were used to compute output weights only for the first output unit. Then the respective roles of the two outputs were reversed, noise was added to the first teacher and weights for the second set of output weights computed. We used a noise fraction of $q = 0.2$. The intuition behind this approach is that when the first set of output weights is learnt, it is learnt from reservoir states in which the effects of feedback from the competing second output are blurred by noise, and vice versa.

Fig. 3 (centre and bottom rows) illustrates the effect of these two strategies. We have not carried out a painstaking optimization of stability under either strategy, and we tested only 20 randomly created ESNs. Therefore, we can only make some qualitative observations:

- Under both strategies roughly half of the 20 runs (8 and 13) resulted in stable generators (with state noise 0.01 for testing stability; a weaker noise might have determined a larger number of stable runs).
- The ridge regression method, when it worked successfully, resulted in more accurate renderings of the target signal than the "noise immunization" method.
- It is not always clear when to classify a trained generator as a successful solution. Deviations from the target come in many forms which apparently reflect different dynamical mechanisms, as can be seen from the examples in the bottom row of Fig. 3. In view of this diversity, criteria for accepting or rejecting a solution will depend on the application context.

To conclude this thread, we give the average training errors and average absolute output weight sizes for the three training setups in the following table:

	Train NRMSE	Stddev	Av. output weights	Stddev
Naive	0.00046	0.00035	0.77	0.48
Regularized	0.0041	0.0019	0.039	0.010
Immunized	0.15	0.050	0.36	0.23

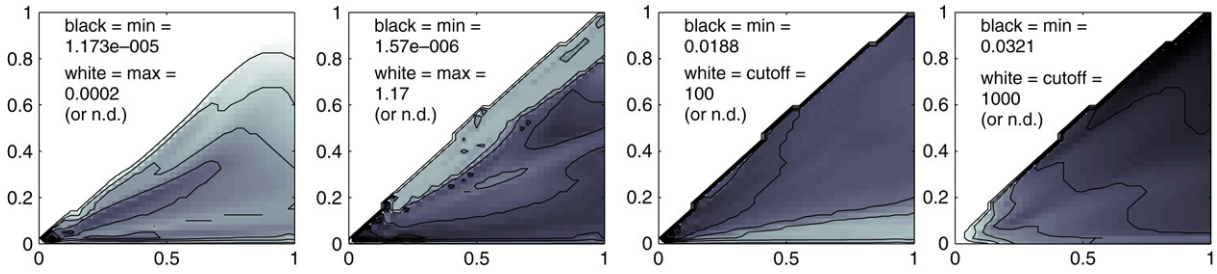

Fig. 5. Various a–ρ-plots, similar to the plots in Section 3, for one of the 20-unit lazy eight ESNs, trained with the "noise immunization" scheme. Other controls were fixed at $s_{\text{in}} = 0.1$, $s_{\text{fb}} = 1.0$. From left to right: (i) training NRMSE, (ii) testing NRMSE, (iii) average absolute output weights, (iv) curvature as in the bottom row of Fig. 2. Grayscale coding reflects logarithmic surface heights.

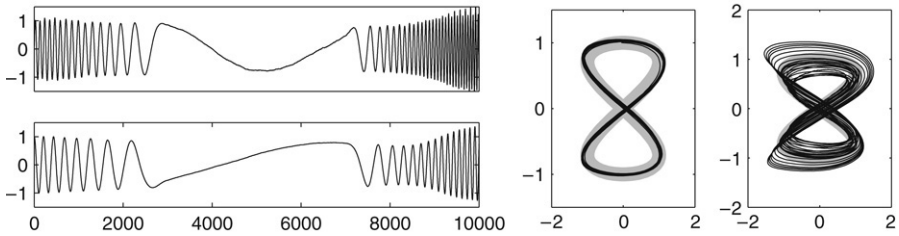

Fig. 6. Lazy eight slowdown/speedup performance. Left: the two generated output signals (x-axis is update cycles). Centre: the output figure eight generated by the original ESN with a constant $\gamma = 1.0$. Right: the 2-dim figure eight plot obtained from the time-warped output signals.

While we know how to quantify output accuracy (by MSE) in a way that lends itself to algorithmic optimization, we do not possess a similar quantifiable criterium for long-term stability which can be computed with comparable ease, enabling us to compute a stochastic gradient online. The control parameter settings $s_{\text{in}} = 0.1, s_{\text{fb}} = 1.0, \rho = a = 0.02$ were decided by a coarse manual search using the "noise immunization" scheme. The optimization criterium here was long-term stability (checked by running trials), not precision of short-term testing productions. While the settings for s_{in} and s_{fb} appeared not to matter too much, variation in ρ and a had important effects; we only came close to anything looking stable once we used very small values for these two parameters. Interestingly, very small values for these parameters also are optimal for small training MSE and testing MSE (the latter computed only from 300-step iterated predictions without state noise to prevent the inherent instabilities to become prominent). Fig. 5 illustrates how the training and testing error and related quantities vary with a and ρ. The main observations are that (i) throughout almost the entire a–ρ-range, small to very small NRMSEs are obtained; (ii) a clear optimimum in training and testing accuracy is located close to zero values of a and ρ — thus the values $\rho = a = 0.02$ which were found by aiming for stability also reflect high accuracy settings; (iii) the most accurate values of a and ρ correspond to the smallest output weights, and (iv) the curvature which is an impediment for the gradient-descent optimization schemes from Section 3 rises to high values as a and ρ approach their optimum region — which would effectivly preclude using gradient descent for localizing these optima. This combination of small output weights, high accuracy and at the same time, high curvature, is at odds with the intuitions we developed in Section 3, calling for further research.

Affording the convenience of a time scale parameter γ, we finally tried to speed up and slow down one of the "noise immunization" trained networks. Fig. 6 (left) shows the x and y outputs obtained in a test run (with again a teacher-forced ramp-up of length 1000, not plotted) where the original $\gamma = 1$ was first slowly reduced to $\gamma = 0.02$, then held at that value for steps $3000 \leq n \leq 7000$, then again increased to $\gamma = 1.0$ at $n = 8500$ and finally to $\gamma = 2.0$ at $n = 10{,}000$. The slowdown and speedup works in principle, but the accuracy of the resulting pattern suffers, as can be seen in Fig. 6 (right). Speeding up beyond $\gamma = 3.0$ made the network lose any visible track of the figure 8 attractor altogether (not shown).

An inspection of Fig. 6 reveals that the amplitude of the output signals diminishes when γ drops, and rises again with γ beyond the correct amplitude. Our explanation rests on a generic phenomenon in discretizing dynamics with the Euler approximation: if the stepsize is increased, curvature of the approximated trajectories decreases. In the case of our ESNs, this means that with larger gains γ (equivalent to larger Euler stepsize) a reservoir neuron's trajectory will flatten out, and so will the generated network output, i.e. oscillation amplitudes grow. Using higher-order methods for discretizing the network dynamics would be a remedy, but the additional implementation and computational cost may make this unattractive.

5. The "Japanese Vowels" dataset

5.1. Data and task description

The "Japanese Vowels" (JV) dataset[1] is a frequently used benchmark for time series classification. The data record

[1] Obtainable from http://kdd.ics.uci.edu/, donated by Mineichi Kudo, Jun Toyama and Masaru Shimbo.

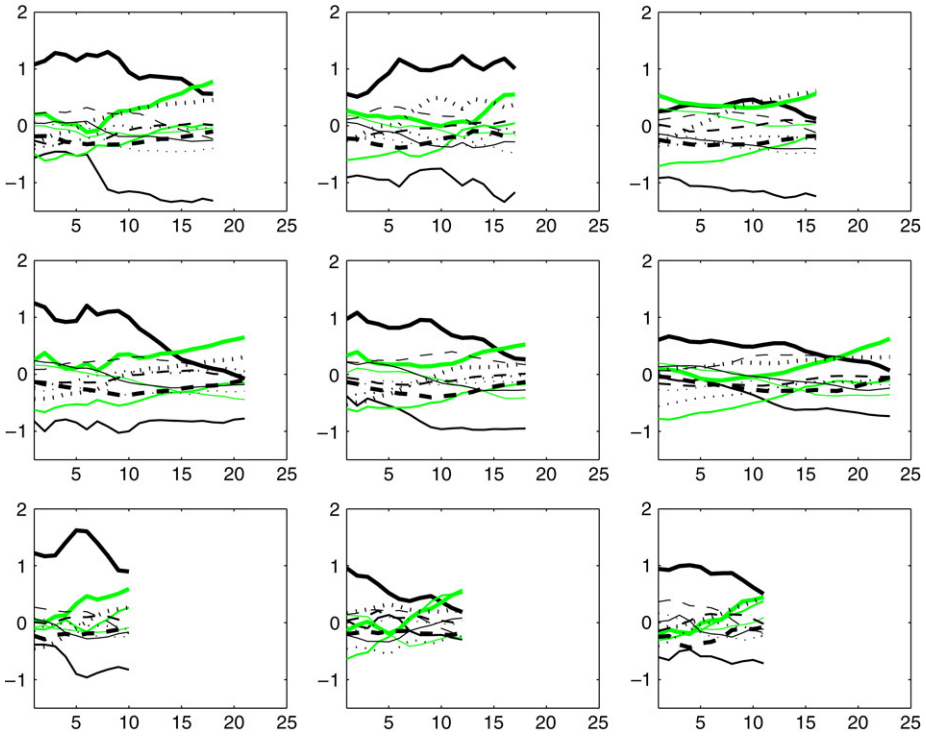

Fig. 7. Illustrative samples from the "Japanese Vowels" dataset. Three recordings from three speakers are shown (one speaker per row). Horizontal axis: discrete time, vertical axis: LPC cepstrum coefficients.

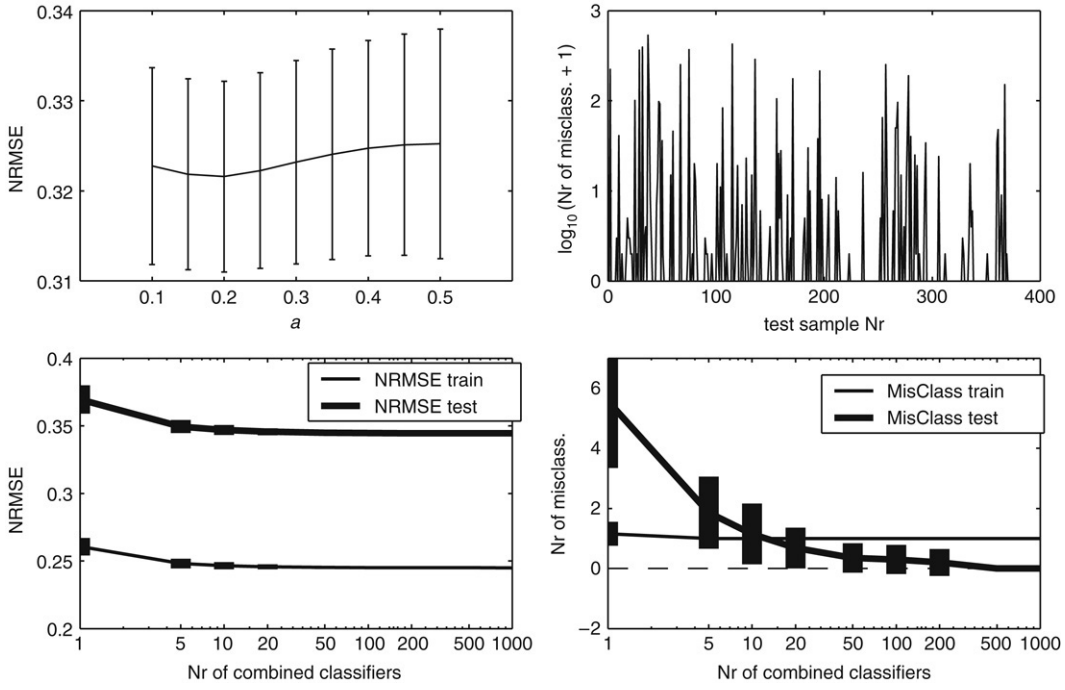

Fig. 8. Diagnostics of the JV task. Top left: cross-validation NRMSE for optimizing the leaking rate a. Top right: Distribution of misclassified test samples, summed over 1000 individual nets. y-scaling is logarithmic to base e. Bottom left: average NRMSE for varying Nrs of combined ESNs. Error bars indicate stddev. Bottom right: Average test misclassifications. Again, error bars indicate stddev. For details see text.

utterances of nine Japanese male speakers of the vowel /ae/. Each utterance is represented by 12 LPC cepstrum coefficients. There are 30 utterances per speaker in the training set, totaling to 270 samples, and a total of 370 test samples lengths distributed unevenly over the speakers (ranging from 24 to 88 samples). The utterance sample lengths range from 7 to 29. Fig. 7 gives an impression. The task is to classify the speakers of the test samples. Various techniques have been applied to

this problem (Barber, 2003; Geurts, 2001; Kudo, Toyama, & Shimbo, 1999; Strickert, 2004). The last listed obtained the most accurate model known to us, a self-organizing "merge neural gas" neural network with 1000 neurons, yielding an error rate of 1.6% on the test data, which corresponds to six misclassifications.

5.2. Setup of experiment

We devoted a considerable effort to the JV problem. Bringing down the test error rate to zero forced us to sharpen our intuitions about setting up ESNs for classifying isolated short time series. In this subsection we not only describe the setup that we finally found best, but also share our experience about setups that came out as inferior — hoping that this may save other users of ESNs unnecessary work.

5.2.1. Small and sharp or large and soft?

There are numerous reasonable-looking ways how one can linearly transform the information from the input-excited reservoir into outputs representing class hypotheses. We tried out four major variations. To provide notation for a discussion, assume that the ith input sequence has length l_i, resulting in l_i state vectors $\mathbf{s}^i(1), \ldots, \mathbf{s}^i(l_i)$ when the reservoir is driven with this sample sequence. Here $\mathbf{s}^i(n) = [\mathbf{x}^i(n); \mathbf{u}^i(n)]$ is the extended state vector, of dimension $N + K$, obtained by joining the reservoir state vector with the input vector; by default we extend reservoir states with input vectors to feed the output units (see Eq. (4)). From the information contained in the states $\mathbf{s}^i(1), \ldots, \mathbf{s}^i(l_i)$ we need to somehow compute nine class hypotheses $(h^i_m)_{m=1,\ldots,9}$, giving values between 0 and 1 after inputting the ith sample to the ESN. Let $(d^i_m)_{m=1,\ldots,9\ i=1,\ldots,270}$ be the desired training values for the nine output signals, that is, $d^i_m = 1$ if the ith training sample comes from the mth class, and $d^i_m = 0$ otherwise. We tested the following alternatives to compute h^i_m:

1. Use nine output units $(y_m)_{m=1,\ldots,9}$, connect each of them to the input and reservoir units by an $1 \times (N + K)$ sized output weight vector $\mathbf{w}_m$, and compute $\mathbf{w}_m$ by linear regression of the targets (d^i_m) on all $\mathbf{s}^i(n)$, where $i = 1, \ldots, 270; n = 1, \ldots, l_i$. Upon driving the trained ESN with a sequence i of length l_i, average each network output unit signal by putting $h^i_m = (\sum_{1 \le n \le l_i} y_m(n))/l_i$. A refinement of this method, which gave markedly better results in the JV task, is to drop a fixed number of initial states from the extended state sequences used in training and testing, in order to wash out the ESNs initial zero state. — *Advantage*: simple and intuitive; would lend itself to tasks where the target time series are not isolated but embedded in a longer background sequence. *Disadvantage*: Output weights reflect a compromise between early and late time points in a sequence; this will require large ESNs to enable a linear separation of classification-relevant state information from different time points. *Findings:* Best results (of about two test misclassifications) achieved with networks of size 1000 (about 9000 trained parameters) with regularization by state noise (Jaeger, 2002b); no clear difference between standard and leaky integrator ESNs.
2. Like before, but use only the last extended state from each sample. That is, compute $\mathbf{w}_m$ by linear regression of the targets d^i_m on all $\mathbf{s}^i(l_i)$, where $i = 1, \ldots, 270$. Put $h^i_m = y_m(l_i)$. Rationale: due to the ESNs short-term memory capacity (Jaeger, 2002a), there is hope that the last extended state incorporates information from earlier points in the sequence. — *Advantage*: even simpler and in tune with the short-term memory functionality of ESNs; would lend itself to usage in embedded target detection and classification (the situation we will encounter in Section 6). *Disadvantage*: when there are time series of markedly different lengths to cope with (which is here the case), the short-term memory behaviour of the ESN would either have to be adjusted to each sample according to its length, or one runs danger of optimal reflection of past information in the last state only for a particular length. *Findings*: the poorest-performing design in this list. The lowest test misclassification numbers we could achieve were about eight with 50-unit, leaky integrator ESNs (about 540 trainable parameters). Adjusting the short-term memory characteristics of ESNs to sequence length by using leaky-integrator ESNs whose gain γ was scaled inversely proportional to the sequence length l_i (such that the ESN ran proportionally slower on longer sequences) only slightly improved the training and testing classification errors.
3. Choose a small integer D, partition each state sequence $\mathbf{s}^i(n)$ into D subsequences of equal length, take the last state $\mathbf{s}^i(n_j)$ from each of the D subsequences ($n_j = j * l_i/D$, $j = 1, \ldots, D$, interpolate between states if D is no divisor of l_i). This gives D extended states per sample, which reflect a few equally spaced snapshots of the state development when the ESN reads a sample. Compose D collections of states for computing D sets of regression weights, where the jth collection ($j = 1, \ldots, D$) contains the states $\mathbf{s}^i(n_j)$, $i = 1, \ldots, 270$. Use these regression weights to obtain for each test sample D auxiliary hypotheses h^i_{mj}, where $m = 1, \ldots, 9; \quad j = 1, \ldots, D$. Each h^i_{mj} is the vote of the "jth section expert" for class m. Combine these into h^i_m by averaging over j. *Advantage*: even small networks (order of 20 units) are capable of achieving zero training error, so we can hope for good models with rather few trained parameters (e.g., $D * (N + K) * 9 \approx 900$ for $D = 3, N = 20$). This seems promising. *Disadvantage*: Not transferable to embedded target sequences. Prone to overfitting. *Findings*: This is the method by which we were first able to achieve zero training error easily. We did not explore this method in more detail but quickly went to the technique described next, which is even sharper in the sense of needing still fewer trainable parameters for zero training error.
4. Use D segment-end-states $\mathbf{s}^i(n_j)$ per sample, as before. For each training sample i, concatenate them into a single joined state vector $\mathbf{s}^i = [\mathbf{s}^i(n_1); \ldots; \mathbf{s}^i(n_D)]$. Use these 270 vectors to compute 9 regression weight vectors for the 9 classes, which directly give the h^i_m values. *Advantage*: Even smaller networks (order of 10 units) are capable of achieving zero

training error, so we can get very small models which are sharp on the training set. *Disadvantage*: Same as before. *Findings*: Because this design is so sharp on the training set, but runs into overfitting, we trimmed the reservoir size down to a mere 4 units. These tiny networks (number of trainable df's: $D * (N + K) * 9 \approx 460$ for $D = 3$, $N = 4$) achieved the best test error rate under this design option, with a bit less than six misclassifications, which amounts to the current best from the literature (Strickert, 2004). Standard ESNs under this design yielded about 1.5 times more test misclassifications than leaky-integrator ESNs. We attribute this to the in-built smoothing afforded by the integration (check Fig. 7 to appreciate the jitter in the data), and to the slow timescale (relative to sample length) of the dominating feature dynamics, to which leaky integrator ESNs can adapt better than standard ESNs. Although the performance did not come close to the two misclassifications we obtained with the first strategy in this list, we found this design very promising because it worked already very well with tiny networks. However, something was still missing.

5.2.2. Combining classifiers

We spent much time on optimizing single ESNs in various setups, but never were able to consistently get better than two misclassifications. The breakthrough came when we joined several of the "design No. 4" networks into a combined classifier.

There are many reasonable ways of how the class hypotheses generated by individual classifiers may be combined. Following the tentative advice given in Duin (2002), Kittler, Hatef, Duin, and Matas (1998), we opted for the mean of the individual votes. This combination method has been found to work well in the cited literature in cases where the individual classifiers are of the same type and are weak, in the sense of exploiting only (random) subspace information from the input data. In this scenario, vote combination by taking the mean has been suggested because (our interpretation) it averages out the vote fluctuations that are due to the single classifiers' biases, — hoping that because the individual classifiers are of the same type, but are basing their hypotheses on individually, randomly constituted features, the "vote cloud" centers close to the correct hypothesis.

Specifically, we used randomly created "design No. 4" leaky-integrator ESNs with four neurons and $D = 3$ as individual classifiers. With only four neurons these nets will have markedly different dynamic properties across their random instantiations (which is not the case for larger ESNs where internetwork differences become insignificant with growing network size). Thus each reservoir can be considered a detector for some random, four-dimensional, nonlinear, dynamic feature of the input signals. Joining $D = 3$ extended state vectors into $\mathbf{s}^i$, as described for design No. 4, makes each four-neuron ESN transform an input sequence into a static $(4 + K) * D$-dimensional feature vector, where the contribution from the $4 * D$ reservoir state components varies across network instantiations. Training the output weights of such a network amounts to attempting to linearly separate the nine classes in the $(4 + K) * D$-dimensional feature space.

Fig. 8 (top right panel) indicates which of the 370 test sequences have been misclassified by how many from among 1000 random "design No. 4" leaky-integrator ESNs. The figure illustrates that many of the samples got misclassified by one or several of the 1000 ESNs, justifying our view that here we have weak classifiers which rely on different subsets of sample properties for their classification.

5.2.3. Data preparation

We slightly transformed the raw JV data, by subtracting from each of the 12 inputs the minimum value this channel took throughout all training samples. We did so because some of the inputs had a considerable systematic offset. Feeding such a signal in its raw version would amount to adding a strong bias constant to this channel, which would effectively shift the sigmoids f of reservoir units away from their centered position toward their saturation range.

In addition to the 12 cepstrum coefficients, we always fed a constant bias input of size 0.1. Furthermore, the input from a training or testing sequence i was augmented by one further constant $C_{l_i} = l_i / l_{\text{maxTrain}}$, where l_{maxTrain} was the maximum among all sequence lengths in the training set. This input is indicative of the length of the ith sequence, a piece of information that is rather characteristic of some of the classes (compare Fig. 7). Altogether this gave us 14-dimensional input vectors.

5.2.4. Optimizing global parameters

We used the network update equation (11) for this case study and thus had to optimize the global parameters N, a, ρ, s^{in}, and s^{ν}. Because our gradient descent optimization does not yet cover nonstationary, short time series, we had to resort to manual experimentation to optimize these. To tell the truth, we relied very much on our general confidence that ESN performance is quite robust with respect to changes in global control parameters, and invested some care only in the optimization of the leaking rate a, whereas the other parameters were settled in a rather offhand fashion.

To use a network size of $N = 4$ was an ad-hoc decision after observing that individual nets with $N = 10$ exhibited significant overfitting. We did not try any other network size below 10.

The input scaling was kept at a value of $s^{\text{in}} = 1.5$ from the preliminary experiments which we briefly pointed out before. Likewise, the spectral radius was just kept from our preliminary attempts with single-ESN models at a value of $\rho = 0.2$. We did not insert any state noise (that is, we used $s^{\nu} = 0$) because stability was not at stake in the absence of output feedback; because the data were noisy in themselves; and because the output weights we obtained were moderately sized (typical absolute sizes averaged between 1 and 10, with a large variation between individual ESNs).

In order to find a good value for a, we carried out a tenfold cross-validation on the training data for various values of a, for 25 independently created four-unit ESNs. Fig. 8 (top left) shows

the findings for $a = 0.1, 0.15, \ldots, 0.5$. Error bars indicate the NRMSE standard deviation across the 25 individual ESNs. This outcome made us choose a value of $a = 0.2$. As we will see, this NRMSE is very nonlinearly connected to the ultimate classification performance, and small improvements in NRMSE may give a large benefit in classification accuracy.

5.2.5. Details of network configuration

We used output units with the tanh squashing function. We linearly transformed the 0-1-teacher values by $0 \mapsto -0.8$, $1 \mapsto 0.8$ to exploit the range of this squashing function (for NRMSE calculations and plots we undid this transformation wherever network output values were concerned). The reservoir and input connection weight matrices were fully connected, with entries drawn from a uniform distribution over $[-1, 1]$. The reservoir weight matrix $\mathbf{W}$ was subsequently scaled to a spectral radius of 1, to be scaled by $\rho = 0.2$ in our update equation (11). In each run of a network with a training or testing sample, the initial network state was zero.

5.3. Results

We trained 1000 four-neuron ESNs individually on the training data, then partitioned the trained networks into sets of size 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, obtaining 1000 individual classifiers, 500 classifiers which combined two ESNs, etc. up to one combined classifier made from all 1000 ESNs.

Fig. 8 comprises the performance we observed for these classifiers. The average number of test misclassifications for individual ESNs was about 5.4 with a surprisingly small standard deviation. It dropped below 1.0 when 20 ESNs were combined, then steadily went down further until zero misclassifications was reached by the collectives sized 500 and 1000. The mean training misclassification number was always exactly one except for the individual ESNs. An inspection of data revealed that it was always the same training sample (incidentally, the last of the 270) which was misclassified.

6. Time warping invariant echo state networks

Time warping of input patterns is a common problem when recognizing human generated input or dealing with data artificially transformed into time series. The most widely used technique for dealing with time-warped patterns is probably dynamic time warping (DTW) (Itakura, 1975) and its modifications. It is based on finding the cheapest (w.r.t. some cost function) mapping between the observed signal and a prototype pattern. The price of the mapping is then taken as a classification criterion. Another common approach to time-warped recognition uses hidden Markov models (HMMs) (Rabiner, 1990). The core idea here is that HMMs can "idle" in a hidden state and thus adapt to changes in the speed of an input signal. An interesting approach of combining HMMs and neural networks is proposed in Levin, Pieraccini, and Bocchieri (1992), where neurons that time-warp their input to match it to its weights optimally are introduced.

A simple way of directly applying RNNs for time-warped time series classification was presented in Sun, Chen, and Lee (1993). We take up the basic idea from that work to derive an effective method for dynamic recognition of time-warped patterns, based on leaky-integrator ESNs.

6.1. Time warping invariance

Intuitively, time warping can be understood as variations in the "speed" of a process. For discrete-time signals obtained by sampling from a continuous-time series it can alternatively be understood as variation in the sampling rate. By definition two signals $\alpha(t)$ and $\beta(t)$ are connected by an approximate continuous *time warping* (τ_1, τ_2), if τ_1, τ_2 are strictly increasing functions on $[0, T]$, and $\alpha(\tau_1(t)) \cong \beta(\tau_2(t))$ for $0 \le t \le T$. We can choose one signal, say $\alpha(t)$, as a reference and all signals that are connected with it by some time warping (e.g. $\beta(t)$) call *(time-)warped* versions of $\alpha(t)$. We will also refer to a time warping (τ_1, τ_2) as a single *time warping (function)* $\tau(t) = \tau_2(\tau_1^{-1}(t))$ which connects the two time series by $\beta(t) = \alpha(\tau(t))$.

We will deal with the problem of recognition (detection plus classification) of time-warped patterns in a background signal, and follow the approach originally proposed in Sun et al. (1993). This method does not look for a time warping that could map an observed signal to a target pattern. Time-warping invariance is achieved by normalizing time dependence of the state variables with respect to the length of trajectory of the input signal in its phase space. In other words, the input signal is considered in a "pseudo-time" domain, where "time span" between two subsequent pseudo time steps is proportional to the metric distance in the input signal between these time steps. As a consequence, input signals will be changing with a constant metric rate in this "pseudo-time" domain. In continuous time, for a k-dimensional input signal $\mathbf{u}(t)$, $\mathbf{u} : \mathbb{R}^+ \to \mathbb{R}^k$ we can define such a time warping $\tau'_{\mathbf{u}} : \mathbb{R}^+ \to \mathbb{R}^+$ by

$$\mathrm{d}\tau'_{\mathbf{u}}(t)/\mathrm{d}t = b \cdot \|\mathrm{d}\mathbf{u}(t)/\mathrm{d}t\|, \tag{28}$$

where b is a constant factor. Note, that the time-warping function $\tau'_{\mathbf{u}}$ depends on the signal $\mathbf{u}$ which it is warping. Then the signal warped by $\tau'_{\mathbf{u}}$ (i.e. in the "pseudo-time" domain) becomes $\mathbf{u}(\tau'_{\mathbf{u}}(t))$, and as a consequence $\|\mathrm{d}\mathbf{u}(\tau'_{\mathbf{u}}(t))/\mathrm{d}t\| = 1/b$, i.e. the k-dimensional input vector $\mathbf{u}(\tau'_u(t))$ changes with a constant metric rate equal to $1/b$ in this domain. Furthermore, if two signals $\mathbf{u}_1(t)$ and $\mathbf{u}_2(t)$ are connected by a time warping τ, then time warping them with $\tau'_{\mathbf{u}_1}$ and $\tau'_{\mathbf{u}_2}$ respectively results in $\mathbf{u}_1(\tau'_{\mathbf{u}_1}(t)) = \mathbf{u}_2(\tau'_{\mathbf{u}_2}(t))$, which is what we mean by *time-warping invariance* (see Fig. 9 for the graphical interpretation of the $k = 1$ case).

A continuous time-processing device governed by an ODE with a time constant c could be made invariant against time warping in its input signal $\mathbf{u}(t)$, if for any given input it could vary its processing speed according to $\tau'_{\mathbf{u}}(t)$ by changing the time constant c in the equations describing its dynamics. This is an alternative to time un-warping the input signal $\mathbf{u}(t)$ itself. Leaky-integrator neurons offer their service at this

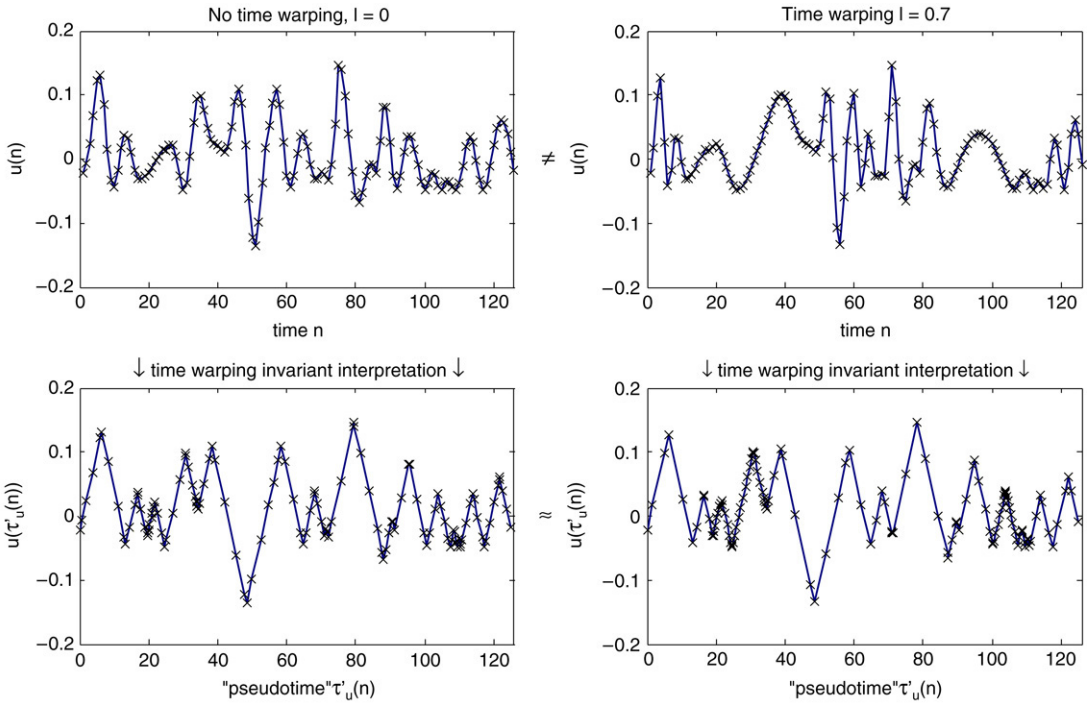

Fig. 9. A time-warping invariant interpretation of two one-dimensional signals connected by time warping. Top left: reference signal with $l = 0$. Top right: a version of the same with $l = 0.7$. Bottom panels: transformations from the top panel sequences into the "pseudotime" domain $\tau'_u(n)$ according to (28). We can observe that for one-dimensional signals the transformation to warping-invariant pseudo-time causes a significant loss of information — the signal $u(\tau'_u(n))$ can be fully described by the sequence of values at local minimums and maximums of $u(n)$.

point. Considering first the continuous-time version of leaky-integrator neurons (Eq. (1)), we would use the inverse time constant $1/c$ to speed up and slow down the network, by making it time varying according to (28). Concretely, we would set

$$c(t) = (b \cdot \|\mathrm{d}\mathbf{u}(t)/\mathrm{d}t\|)^{-1}. \tag{29}$$

If we have discrete sampled input $\mathbf{u}(n)$, we use the discrete-time approximation from Eq. (6) and make the gain γ time varying by

$$\gamma(n) = b \cdot \|\mathbf{u}(n) - \mathbf{u}(n-1)\|. \tag{30}$$

6.2. Data, learning task, and ESN setup

We prepared continuous-time time series of various dimensions, consisting of a red-noise background signal into which many copies of a short target pattern were smoothly embedded. The task is to recognize the target pattern in the presence of various degrees of time warping applied to the input sequences.

Specifically, a red-noise background signal $\mathbf{g}(t)$ was obtained by filtering out 60% of the higher frequencies from a $[-0.5, 0.5]^k$ uniformly distributed white-noise signal. A short target sequence $\mathbf{p}(t)$ of duration T_p with a similar frequency makeup was generated and smoothly embedded into $\mathbf{g}(t)$ at random locations, using suitable soft windowing techniques for the embedding to make sure that no novel (high) frequency components were created in the process (details in Lukoševičius et al. (2006)). This embedding yielded

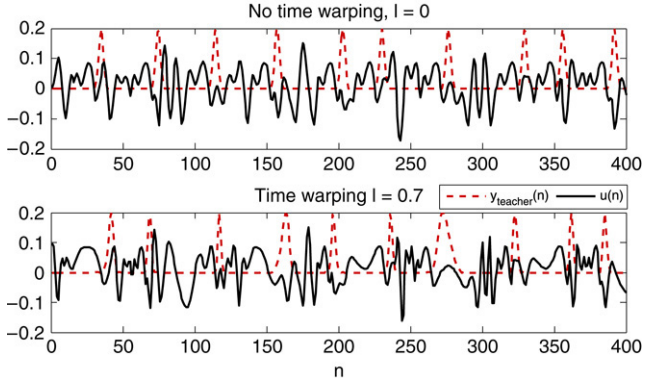

Fig. 10. A fragment of a one-dimensional input and corresponding teacher signal without and with time warping.

continuous-time input signals $\mathbf{u}(t)$ which were subsequently sampled and time warped to obtain discrete-time inputs $\mathbf{u}(n)$.

The (one-dimensional) desired output signal $d(t)$ was constructed by placing narrow Gaussians on a background zero signal, at the times where the target patterns end. The height of the Gaussians is scaled to one. Fig. 10 gives an impression of our data.

Discrete time time-warped observations $\mathbf{u}(n)$ of the process $\mathbf{u}(t)$ were drawn as $\mathbf{u}(n) = \mathbf{u}(\tau(n))$, where $\tau : \mathbb{N} \to \mathbb{R}^+$ fulfilled both time warping and discretization functions. Both $\mathbf{u}(t)$ and the corresponding teacher $d(t)$ were discretized/time warped together. More specifically, we used

$$\tau(n) = (n + 10 \cdot l \sin(0.1n)), \tag{31}$$

where $l \in [0, 1]$ is the *level* of time warping: $\tau(n)$ is a straight line (no time warping) when $l = 0$, and is a nondecreasing function as long as $l \leq 1$. In the extreme case of $l = 1$, time "stands still" at some points in the trajectory. In the obtained signals $\mathbf{u}(n)$ the pattern duration T_p on average corresponded to 20 time steps.

The learning task for the ESN is to predict $d(n)$ from $\mathbf{u}(n)$. In each network-training-testing trial, the warping level l on the training data was the same as on the test data. Training sequences were of length 3000, of which the first 1000 were discarded before computing the output weights in order to wash out initial network transients. For testing, sequences of length 500 were used throughout.

We used leaky-integrator ESNs with 50 units. The reservoir weight matrix was randomly created with a connectivity of 20%; nonzero weights were drawn from a uniform distribution centred on 0 and then globally rescaled to yield a reservoir weight matrix $\mathbf{W}$ with unit spectral radius. We augmented inputs $\mathbf{u}(n)$ by an additional constant bias signal of size 0.01, yielding a $K = k + 1$ dimensional input overall. Input weights were randomly sampled from a uniform distribution over $[-1, 1]$. There were no output feedbacks.

The spectral radius ρ and the leaking rate a were manually optimized to values of $\rho = 0.3$, $a = 0.3$ on inputs without time warping. Likewise, on data without time warping we determined by manual search an optimal reference value for the gain of $\gamma_0 = 1.2$.

Turning to time-warped data, for each time series we set the constant b from (28) such that on average a gain of $\langle \gamma(n) \rangle = 1.2$ would result from running the ESN with a dynamic gain according to (30), that is, for a given input sequence we put

$$b = 1.2/\langle \|\mathbf{u}(n) - \mathbf{u}(n-1)\| \rangle. \tag{32}$$

This results in the following update equation:

$$\mathbf{x}(n+1) = (1 - a\gamma(n))\, \mathbf{x}(n) + \gamma(n)\, f(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n)), \tag{33}$$

where $\gamma(n) = b \cdot \|\mathbf{u}(n) - \mathbf{u}(n-1)\|$. As a safety catch, the range of $\gamma(n)$ was bounded by a hard limit to satisfy the constraint from Eq. (5), that is, $\gamma(n)$ was limited to values of at most $1/a$.

6.3. Results

We compared the performance of three types of ESNs: (i) leaky-integrator ESNs with fixed $\gamma = 1.2$; (ii) time-warping invariant ESNs (TWIESNs) according to Eq. (28); and for comparison we also constructed (iii) a version of leaky-integrator ESNs which unwarped the signals in an optimal way, using the information (which in real life is unavailable) of the time warping τ: this ESN version has $\gamma(n)$ externally set to $\min(b[\tau(n) - \tau(n-1)], 1/a)$. Let us call this type of networks, "oracle" ESNs.

For each run, we computed two performance criteria: the NRMSE of network output vs. $d(n)$, and the ratio of q of correctly recognized targets. To determine q, each connected interval of n for which $y(n) > h$ corresponds to one claimed target recognition. Any such event is considered a correct recognition if the intervals of n where $y(n) > h$ and $y_{\text{teacher}}(n) > h$ overlap. The value of h was optimized by maximizing q over the training data. Then, q is evaluated as *Nr. of correct recognitions*/(*Nr. of targets in test sequence* + *Nr. of claimed recognitions* − *Nr. of correct recognitions*). Fig. 11 summarizes the findings for $k = 1, 3, 10, 30$. Each plot renders the average values from 100 runs with independently created ESNs.

The main observations are the following:

- If k and l are small, the fixed leaky integrator ESNs do better than TWIESNs. This can be explained by the loss of temporal information induced by the transformation to pseudotime.
- If k or l is large, TWIESNs outperform fixed leaky-integrator ESNs, and indeed come close to oracle ESNs when k is 10 or larger. This means that the purely temporal information contributes little to the recognizability of patterns, as opposed to the "structural" (time-scaling invariant) information in the input sequences. Since in real-life applications where time warping is an issue (for instance, speech or handwriting recognition), inputs are typically feature vectors of sizes beyond 10, our approach seems to hold some promise.
- NRMSE and recognition rate q remain almost, but not quite constant for TWIESNs as l varies from zero to one. Naively, one should expect that q is not affected at all by l, because TWIESNs should "see" indentical pseudo-time series across all l, as in the bottom panels of Fig. 9. Indeed, this is not exactly the case: when $\gamma(n)$ becomes large, the reservoir dynamics is not an exact sped-up version of a slower dynamics, due to inaccuracies of the Euler discretization. This explains the systematic, slight increase in NRMSE as l grows. A remedy would be to use higher-order discrete approximations of the continuous-time leaky-integrator dynamics from (1), for instance Runge–Kutta approximations.

7. Conclusion

Leaky-integrator ESNs are only slightly more complicated to implement and to use than standard ESNs and appear to us as quite flexible devices when timescale phenomena are involved, where standard ESNs run into difficulties. Caution is, however, advised when simple Euler approximations to the continuous-time leaky- integrator dynamics are used.

Two questions were encountered which we consider to be of longer-lasting importance:

- Find computationally efficient ways to optimize the global scaling parameters of ESNs. Specifically, naive stochastic gradient descent approaches, as introduced in this paper, suffer from poor stability properties, which renders them impractical in many cases.
- Analyse conditions and provide practical training schemes that ensure stability of ESNs trained as complex pattern generators.

Finally we want to point out a question which we think would concern *any* modelling approach to nonlinear dynamics,

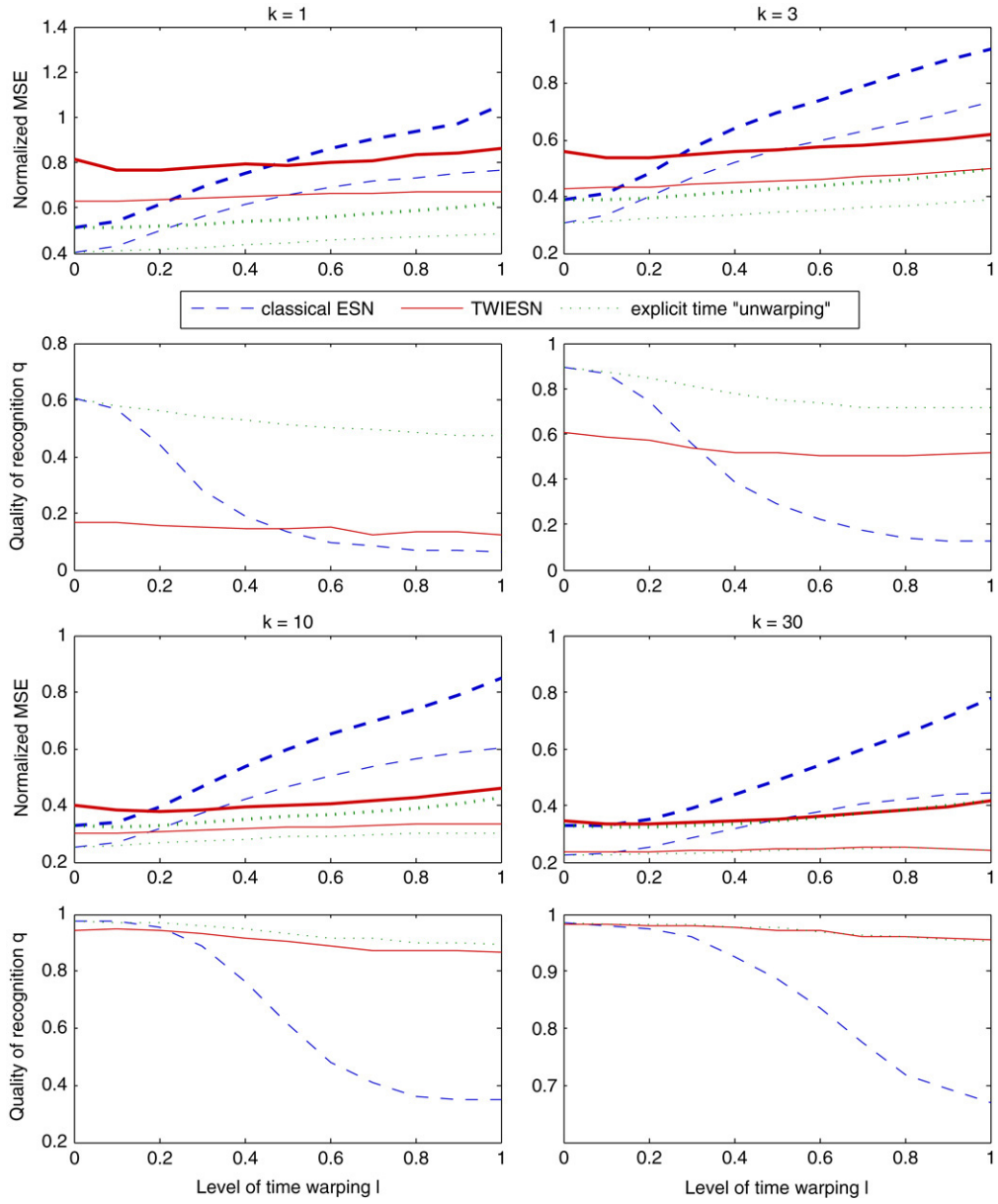

Fig. 11. Performance of fixed-γ ESNs (dashed), oracle ESNs (dotted) and TWIESNs (solid), for input dimensions $k = 1, 3, 10$ and 30, and different levels of time warping l (x-axes). Bold lines: test performance, thin lines: performance on training data. For details see text.

whether based on RNNs or not. Humans can to some extent adjust the speed (and amplitude) of generated motor (or vocal) trajectories. For instance, they can speak faster or slower, or perform a gesture at different velocity. What could an underlying neural mechanism be? We can't think of a biologically plausible mechanism by which biological brains could dynamically adjust the physical time constants of their neurons in a quick, adaptive and homogeneous fashion, in the way we can adjust γ in our equations. However, biological brains can apparently adjust gains of neural connection pathways. It would be an interesting challenge for theoretical neuroscience to devise dynamic or RNN mechanisms by which trajectories can be sped up or slowed down, not by directly changing time constants but instead by changing connection (coupling) gains.

Acknowledgments

The work on time-warping invariant ESNs reported here was supported by student contract grants for ML and DP from Planet intelligent systems GmbH, Raben Steinfeld, Germany. The authors would also like to thank five (!) anonymous reviewers of the NIPS 2005 conference, who helped to improve the presentation of Section 6, which once was a NIPS submission. The treatment of the lazy eight task owes much to discussions with J. Steil, R. Linsker, J. Principe and B. Schrauwen. The authors also express their gratitude toward two anonymous reviewers of the Neural Networks Journal for detailed and helpful commentaries. International patents for the basic ESN architecture and training algorithms have been claimed by Fraunhofer IAIS www.iais.fraunhofer.de.

Appendix. Proof of Proposition 1

For any two states $\mathbf{x}(n), \mathbf{x}'(n)$ the following holds:

$$\begin{aligned}
\|\mathbf{x}(n) - \mathbf{x}'(n)\| &= \Big\| \left(1 - a\frac{\delta}{c}\right)(\mathbf{x}(n) - \mathbf{x}'(n)) \\
&\quad + \frac{\delta}{c}(f(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n)) \\
&\quad - f(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}'(n)))\Big\| \\
&\leq \left(1 - a\frac{\delta}{c}\right)\|\mathbf{x}(n) - \mathbf{x}'(n)\| \\
&\quad + \frac{\delta}{c}\|f(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) \\
&\quad + \mathbf{W}\mathbf{x}(n)) - f(\mathbf{W}^{\text{in}}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}'(n))\| \\
&\leq \left(1 - a\frac{\delta}{c}\right)\|\mathbf{x}(n) - \mathbf{x}'(n)\| \\
&\quad + \frac{\delta}{c}\|\mathbf{W}\mathbf{x}(n) - \mathbf{W}\mathbf{x}'(n)\| \\
&\leq \left(1 - \frac{\delta}{c}(a - \sigma_{\max})\right)\|\mathbf{x}(n) - \mathbf{x}'(n)\|.
\end{aligned}$$

Thus, $|1 - \frac{\delta}{c}(a - \sigma_{\max})|$ is a global Lipschitz rate by which any two states approach each other in a network update.

References

Barber, D. (2003). Dynamic Bayesian networks with deterministic latent tables. In *Proc. NIPS 2003*. http://www.anc.ed.ac.uk/~dbarber/publications/barber_nips_dethidden.pdf.

Buehner, M., & Young, P. (2006). A tighter bound for the echo state property. *IEEE Transactions on Neural Networks*, *17*(3), 820–824.

Buonomano, D. V. (2005). A learning rule for the emergence of stable dynamics and timing in recurrent networks. *Journal of Neurophysiology*, *94*, 2275–2283. http://www.neurobio.ucla.edu/~dbuono/BuonoJNphy05.pdf.

Duin, R. P. W. (2002). The combining classifier: To train or not to train? In R. Kasturi, D. Laurendeau, & C. Suen (Eds.). *Proceedings 16th international conference on pattern recognition (ICPR16): Vol. II* (pp. 765–770). IEEE Computer Society Press. http://ict.ewi.tudelft.nl/~duin/papers/icpr_02_trainedcc.pdf.

Farhang-Boroujeny, B. (1998). *Adaptive filters: Theory and applications*. Wiley.

Geurts, P. (2001). Pattern extraction for time series classification. In L. De Raedt, A. Siebes (Eds.), *Proc. PKDD 2001* (pp. 115–127). http://www.montefiore.ulg.ac.be/~geurts/publications/geurts-pkdd2001.pdf.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Springer Verlag.

Hinder, M. R., & Milner, T. E. (2003). The case for an internal dynamics model versus equilibrium point control in human movement. *Journal of Physiology*, *549*(3), 953–963. http://www.sfu.ca/~tmilner/953.pdf.

Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *23*(1), 67–72.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks. *GMD report no. 148*. GMD — German National Research Institute for Computer Science. http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf.

Jaeger, H. (2002a). Short term memory in echo state networks. *GMD-report no. 152*. GMD — German National Research Institute for Computer Science. http://www.faculty.iu-bremen.de/hjaeger/pubs/STMEchoStatesTechRep.pdf.

Jaeger, H. (2002b). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. *GMD report no. 159*. Fraunhofer Institute AIS. http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNTutorial.pdf.

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in neural information processing systems*: *Vol. 15* (pp. 593–600). Cambridge, MA: MIT Press. http://www.faculty.iu-bremen.de/hjaeger/pubs/esn_NIPS02.

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, *304*, 78–80. http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNScience04.pdf.

Kittler, J., Hatef, M., Duin, R., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(3), 226–239. http://ict.ewi.tudelft.nl/~duin/papers/pami_98_ccomb.pdf.

Kudo, M., Toyama, J., & Shimbo, M. (1999). Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, *20*(11), 1103–1111.

Levin, E., Pieraccini, R., & Bocchieri, E. (1992). *Advances in neural information processing systems*: *Vol. 4. Time-warping network: A hybrid framework for speech recognition [NIPS conference]* (pp. 151–158).

Lukoševičius, M., Popovici, D., Jaeger, H., & Siewert, U. (2006). Time warping invariant echo state networks. *IUB technical report no. 2*. International University Bremen. http://www.iu-bremen.de/imperia/md/content/groups/research/techreports/twiesn_iubtechreport.pdf.

Maass, W., Joshi, P., & Sontag, E. (2006). Computational aspects of feedback in neural circuits. *PLOS Computational Biology* 31(3), e165, in press (doi:10.1371/journal.pcbi.0020165). http://www.igi.tugraz.at/maass/psfiles/168_v6web.pdf.

Maass, W., Natschlaeger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560. http://www.cis.tugraz.at/igi/maass/psfiles/LSM-v106.pdf.

Mauk, M., & Buonomano, D. (2004). The neural basis of temporal processing. *Annual Review of Neuroscience*, *27*, 307–340.

Ozturk, M. C., Xu, D., & Principe, J. C. (2007). Analysis and design of echo state networks for function approximation. *Neural Computation*, *19*(1), 111–138.

Pearlmutter, B. (1995). Gradient calculation for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(5), 1212–1228. http://www.bcl.hamilton.ie/~bap/papers/ieee-dynnn-draft.ps.gz.

Rabiner, L. (1990). A tutorial on hidden Markov models and selected applications in speech recognition. In A. Waibel, & K. -F. Lee (Eds.), *Readings in speech recognition* (pp. 267–296). San Mateo: Morgan Kaufmann. Reprinted from *Proceedings of the IEEE 77* (2) (1989) 257–286.

Schiller, U., & Steil, J. J. (2005). Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, *63C*, 5–23. http://www.techfak.uni-bielefeld.de/~jsteil/publications.html.

Schmidhuber, J., Gomez, F., Wierstra, D., & Gagliolo, M. (2007). Training recurrent networks by evolino. *Neural Computation*, *19*(3), 757–779.

Stanley, G., Li, F., & Dan, Y. (1999). Reconstruction of natural scenes from ensemble responses in the lateral genicualate nucleus. *Journal of Neuroscience*, *19*(18), 8036–8042. http://people.deas.harvard.edu/~gstanley/publications/stanley_dan_1999.pdf.

Strickert, M. (2004). Self-organizing neural networks for sequence processing. *Ph.D. thesis*. Univ. of Osnabrück, Dpt. of Computer Science. http://elib.ub.uni-osnabrueck.de/publications/diss/E-Diss384_thesis.pdf.

Sun, G. -Z., Chen, H. -H., & Lee, Y. -C. (1993). *Advances in neural information processing systems*: *Vol. 5. Time warping invariant neural networks [NIPS conference]* (pp. 180–187). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Zant, T. van der, Becanovic, V., Ishii, K., Kobialka, H. -U., & Plöger, P. (2004). Finding good echo state networks to control an underwater robot using evolutionary computations. In *CD-ROM Proc. IAV 2004 — The 5th symposium on intelligent autonomous vehicles*. http://www.ais.fraunhofer.de/~vlatko/Publications/Finding_good_esn_FINAL_PhotoCopyReady.pdf.

Zegers, P., & Sundareshan, M. K. (2003). Trajectory generation and modulation using dynamic neural networks. *IEEE Transactions on Neural Networks*, *14*(3), 520–533.